Towards Inference and Learning in Dynamic Bayesian Networks using Generalized Evidence

Christopher James Langmead*

August 2008
CMU-CS-08-151

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA 15213.
E-mail: cjl@cs.cmu.edu

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

Keywords: Graphical Models, Dynamic Bayesian Networks, Inference, Learning, Decision Procedures, propositional SAT, Model Checking, Temporal Logic

Abstract

This report introduces a novel approach to performing inference and learning in *Dynamic Bayesian Networks* (DBN). The traditional approach to inference and learning in DBNs involves conditioning on one or more finite-length observation sequences. In this report, we consider conditioning on what we will call *generalized evidence*, which consists of a possibly infinite set of behaviors compactly encoded in the form of a formula, ϕ, in *temporal logic*. We then introduce exact algorithms for solving inference problems (i.e., computing $P(\mathbf{X}|\phi)$) and learning problems (i.e., computing $P(\Theta|\phi)$) using techniques from the field of *Model Checking*. The advantage of our approach is that it enables scientists to pose and solve inference and learning problems that cannot be expressed using traditional approaches. The contributions of this report include: (1) the introduction of the inference and learning problems over generalized evidence, (2) exact algorithms for solving these problems for a restricted class of DBNs, and (3) a series of case studies demonstrating the scalability of our approach. We conclude by discussing directions for future research.

1 Introduction

Dynamic Bayesian Networks (DBNs) are a family of probabilistic graphical models for representing stochastic processes. The inference and learning problems in DBNs involve computing posterior distributions over unobserved (aka hidden) variables or parameters, respectively, given evidence. In this context, evidence consists of a finite number of observation sequences. Each sequence reveals the sequential behavior of a subset of the variables in the model. This report seeks to generalize the notion of evidence by describing the abstract behavior of the variables using a formula in temporal logic. We then present exact algorithms for solving the inference and learning problems over generalized evidence in a restricted class of DBNs. This report is intended to lay the groundwork for a more comprehensive set of algorithms that will enable inference and learning over generalized evidence in arbitrary graphical models.

A DBN models a stochastic process $P(\mathbf{X}_t)\ \forall t \in T$, where $\mathbf{X}$ is a set of random variables indexed by t. A typical inference problem in a DBN involves conditioning on an *observation sequence*, $\mathbf{o}_{0:t} = (\mathbf{o}(o), ..., \mathbf{o}(t))$, where $\mathbf{O} \subseteq \mathbf{X}$ and $\mathbf{o}(i)$ is the assignment of $\mathbf{O}$ at time index i. For example, the task of *decoding* involves finding an assignment for the unobserved (aka hidden or latent) variables $\mathbf{h}^*_{0:t} : \mathbf{H} = \mathbf{X} \backslash \mathbf{O}$ that maximizes $P(\mathbf{h}_{0:t}|\mathbf{o}_{0:t})$. Notice that an observation sequence describes exactly one trajectory for the observed variables. In this report, we are interested in conditioning on sets of trajectories. We call this generalized evidence.

The following examples illustrate some of the possible uses for generalized evidence.

1. Suppose we have two observation sequences, $\mathbf{o}^a_{0:t_1}$ and $\mathbf{o}^b_{0:t_2}$. Consider the combined observation $\phi := \mathbf{o}^a_{0:t_1} \vee \mathbf{o}^b_{0:t_2}$ where t_1 may or may not equal t_2. Here, ϕ encodes the notion that we are interested in scenarios where either $\mathbf{o}^a_{0:t_1}$ or $\mathbf{o}^b_{0:t_2}$ happens. We can now consider computing the conditional distribution $P(\mathbf{H}_{0:t}|\phi)$.

2. Alternatively, suppose $\phi := \neg\mathbf{o}_{0:t}$. Here ϕ is a *negative observation sequence* encoding, in essence, a forbidden trajectory through the model. Equivalently, ϕ can be interpreted as *every* trajectory through the model *except* $\mathbf{o}_{0:t}$. We can now imagine decoding over the distribution $P(\mathbf{H}_{0:t}|\phi)$. Of course, we can also construct more complicated examples, such as $\phi := \neg(\mathbf{o}^a_{0:t_1} \vee \mathbf{o}^b_{0:t_2})$.

3. We can also describe negative observations over the hidden variables as well. For example, suppose we are interested in assignments where $h_i(3) \neq 0$; that is, the value of variable h_i at time $t = 3$ is not zero. If we now define $\phi := \mathbf{o}^a_{0:t} \vee \mathbf{o}^b_{0:t} \wedge h_i(3) \neq 0$, we consider a new decoding problem where we compute $\mathbf{h}^*_{0:t}$ that maximizes $P(\mathbf{h}_{0:t}|\phi)$.

4. More generally, we may be interested in conditioning on generic *properties* over the hidden and observed variables. For example, we may only be interested in assignments such that the value of some hidden variable h_i is always less than some value, say c_1, until the value of some other variable h_j is greater than some other value, say c_2. If we now define $\phi := \mathbf{o}^a_{0:t} \vee \mathbf{o}^b_{0:t} \wedge h_i(3) \neq 0 \wedge h_i < c_1\ \mathbf{U}\ h_j > c_2$, we obtain a yet another decoding problem. Here, the expression "$h_i < c_1\ \mathbf{U}\ h_j > c_2$" refers to *all* assignments such that $h_i < c_1$ "until" $h_j > c_2$.

These examples represent just some of the kinds of problems that might be considered if we generalize the notion of evidence. The motivation for this work arises from the fields of Systems and Synthetic Biology where DBNs can be used to model biological processes, and one sometimes needs to answer questions like: "what is the probability that event *A* happens before event *B*?". This is not easy to answer using traditional inference algorithms.

This report has three objectives. The *first* objective is to introduce the notion of a generalized evidence and how to efficiently encode it using formulas in *temporal logic*. The *second* objective is to show that DBNs can be used to *symbolically* encode *Kripke structures*, which are a formalism for representing concurrent processes. Kripke structures are central to the field of *Model Checking*. This observation is significant because it implies that a) Model Checking algorithms can be used to solve inference and learning problems in DBNs, and b) inference and learning algorithms for DBNs might be used to solve Model Checking problems. This report focuses on (a) and thus our third objective is to use introduce exact algorithms from the field of Model Checking for solving the inference and learning problem over generalized evidence in a restricted class of DBNs.

This report is organized as follows: We define DBNs and the (traditional) inference and learning problems in Section 2. Next, we introduce the notion of generalized evidence in Section 3. We then briefly summarize some of the fundamental concepts from the field of Model Checking in Section 4. Our algorithm for solving the inference and learning problems over generalized evidence is presented in Section 5 followed by a series of case-studies in Section 6. We conclude by discussing our results and areas for future work in Section 7.

2 Dynamic Bayesian Networks

Given a probability space $(\Omega, \mathcal{F}, P)$, a stochastic process, $\mathcal{M}$ is a collection $\{\mathbf{X}_t : t \in T\}$ where each $\mathbf{X}(t)$ is a set of random variables indexed by a set T, which we will refer to as time. Here, Ω is the sample space, $\mathcal{F}$ is a σ-algebra on subsets of Ω, and P is a function mapping $\mathcal{F}$ to probabilities. A Dynamic Bayesian Network (DBN) is a pair $(\mathcal{B}, \mathcal{B}_\tau)$ encoding M. Here, B is a Bayesian Network, and B_τ is a two-slice Bayesian Network, which are defined in the following two sections. Informally, $\mathcal{B}$ encodes a prior probability distribution over $\mathbf{X}$, while $\mathcal{B}_\tau$ encodes the dynamics of the process.

2.1 Bayesian Networks

The term "Bayesian Network" refers to a specific subclass of *probabilistic graphical models*, which are a family of graph-theoretic approaches to representing and computing over joint probability distributions. A Bayesian Network is a model of $P(\mathbf{X})$ consisting of a directed acyclic graph, $\mathcal{G} = (V, E)$, and a set of conditional probability distributions, Ψ. Figure 1 shows two small Bayesian Networks.

The sets V and Ψ are each isomorphic to a set of random variables, $\mathbf{X}$. Node v_i is a parent of node v_j if there is a directed edge from v_i to v_j. Each edge reveals a dependency between variables. That is $P(X_i, X_j) \neq P(X_i)P(X_j)$ if $e_{i,j} \in E$. Let $N(v_i) \subseteq V$ be the neighbors of node v_i in the

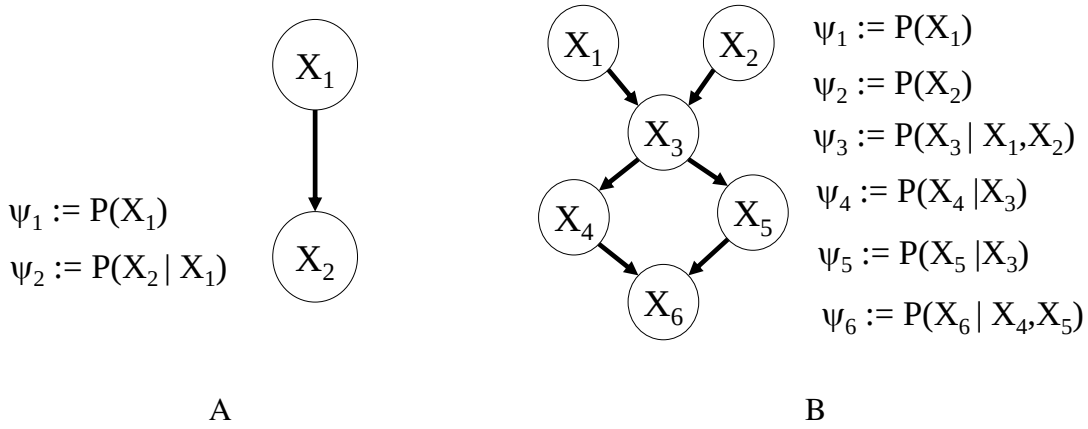


Figure 1: **Probabilistic Graphical Models**: (**A**) A 2-node Bayes Network encoding $P(X_1, X_2)$. (**B**) A 6-node Bayes Net which encodes $P(X_1, X_2, X_3, X_4, X_5, X_6)$.

graph. Let $\mathbf{X}_{N(i)} \subseteq \mathbf{X}$ denote the set of variables corresponding to the neighbors of node v_i. If v_i has parents in the graph then $\psi_i \equiv P(X_i|\mathcal{X}_{N(i)})$ — the conditional probability distribution over X_i, given its parents. Otherwise, $\psi_i \equiv P(X_i)$ — the prior probability distribution over X_i.

The Bayesian Network encodes the joint distribution $P(\mathbf{X})$ in a factored form. Specifically, $P(\mathbf{X}) = \prod_{i=1}^{n} \psi_i = \prod_{i=1}^{n} P(X_i|\mathbf{X}_{N(i)})$. If the graph is sparse, the Bayesian Network is a compact encoding of the joint distribution. For example, suppose $\mathbf{X}$ is a set of 10 Boolean random variables. An explicit encoding of $P(\mathbf{X})$ as a table would require $2^{10} = 1024$ elements. On the other hand, if the joint can be represented using a graph where each node has no more than 3 parents, then the corresponding distribution can be encoded as a Bayesian Network requiring no more than $10 * 2^3 = 80$ elements.

2.2 2-Slice Bayesian Networks

It is also possible to model the dynamics of state transitions using Bayesian Networks. A 2-slice Bayesian Network is a Bayesian Network with $2n$ nodes, encoding the conditional probability distribution $P(\mathbf{X}(t)|\mathbf{X}(t-\delta t))$. Here, $t, \delta t \in \mathbb{R}$ if the 2-slice Bayesian Network describes continuous-time dynamics and $t, \delta t \in \mathbb{Z}$, if the 2-slice Bayesian Network describes discrete-time dynamics. Figure 2-A depicts a 2-slice Bayesian Network. The probability functions (i.e., Ψ) are not shown but they follow the same rules as in the last section; that is, each ψ_i defines a probability distribution over a X_i as a function of the state of the parents of v_i in the graph. The parents of the nodes on the right-hand side are the nodes on the left-hand side, encoding the temporal dependencies. The nodes on the left-hand side do not have parents which means, as before, that the corresponding ψ_i defines a prior probability distribution over X_i.

Naturally, a 2-slice Bayesian network can be "unrolled" to become a k-slice Bayesian Network in order to model longer sequences. Moreover, by unrolling the 2-slice Bayesian Network, we can create more complicated models where the temporal dependencies can span multiple time slices. Figure 2-B, for example, depicts a 4-slice Bayesian Network where the state of variable X_1 at time t is dependent on the state of variables X_1 and X_2 at time $t - \delta t$ *and* on the state of variable X_1 at

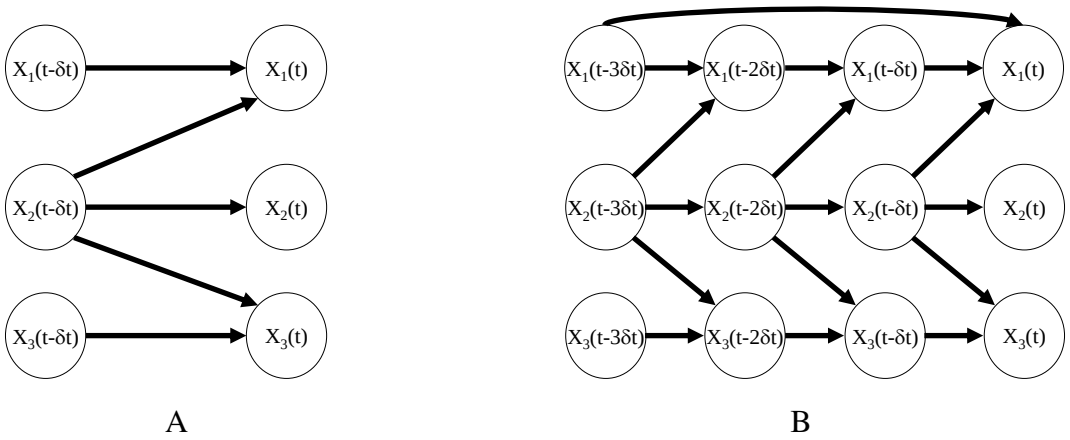


Figure 2: (**A**) This graphical model encodes the distribution $P(\mathbf{X}(t)|\mathbf{X}(t-\delta t))$, given appropriate definition of the potential functions: $\psi_1 \equiv P(X_1(t)|X_1(t-\delta t), X_2(t-\delta t))$, $\psi_2 \equiv P(X_2(t)|X_2(t-\delta t))$, and $\psi_3 \equiv P(X_3(t)|X_2(t-\delta t), X_3(t-\delta t))$. (**B**) This graphical model encodes the distribution $P(\mathbf{X}(t)|\mathbf{X}(t-\delta t), \mathbf{X}(t-3\delta t))$.

time $t - 3\delta t$. The resulting model encodes the distribution $P(\mathbf{X}(t)|\mathbf{X}(t-\delta t), \mathbf{X}(t-3\delta t))$.

2.3 Putting it all together: the Dynamic Bayesian Network

In the previous section, we defined models suitable for defining a prior probability distribution over a set of random variables and one for defining the stochastic dynamics over a set of random variables. We can combine these to create a Dynamic Bayesian Network (DBN), which encodes both a prior and the stochastic dynamics. Figure 3 depicts a DBN. On the left-hand side B is a Bayesian Network encoding $P(\mathbf{X})$ — the prior over $\mathbf{X}$, while B_τ is a 2-slice Bayesian Network encoding $P(\mathbf{X}(t)|\mathbf{X}(t-\delta t))$ — the stochastic dynamics. The dashed edges between B and B_τ define a set of parents for the nodes on the left-hand side of B_τ which, in turn, defines the *joint prior* on $\mathbf{X}$. In contrast, the priors associated with the left-hand nodes in, say, Figure 2 define the marginal priors over each X_i.

2.4 Inference and Learning in DBNs

There are two canonical tasks in graphical models, *inference* and *learning*, both of which are NP-hard. Specifically, inference and learning are exponential in the tree-width of graph. The reader is directed to [14] for more information on traditional inference and learning algorithms for DBNs. One of the goals of this report is to introduce a new approach for posing inference and learning problems. We thus summarize the key aspects of traditional approaches here.

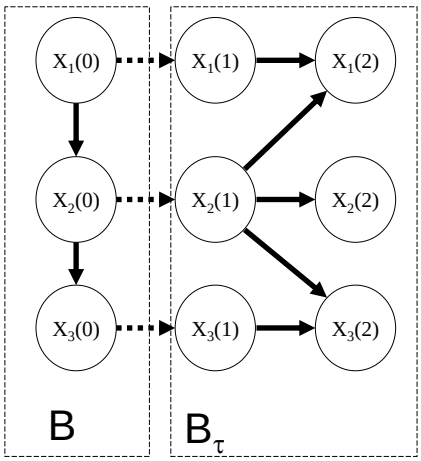


Figure 3: A Dynamic Bayesian Network (DBN). A DBN is really a combination of two graphical models, B, and B_τ. The first model, B, encodes a prior distribution over $\mathbf{X}$. The second model, B_τ, encodes the dynamics of the model.

2.4.1 Evidence

An *observation sequence* $\mathbf{o}_{0:T} = (\mathbf{o}(0), \mathbf{o}(1), ..., \mathbf{o}(T))$ where $\mathbf{O} \subseteq \mathbf{X}$ is an arbitrary subset of $\mathbf{X}$ representing the observed variables. Let $\mathbf{H} = \mathbf{X} \backslash \mathbf{O}$ be the complement of $\mathbf{O}$ (aka the hidden or latent variables). The inference problem in a DBN involves computing $P(\mathbf{H}_{0:T}|\mathbf{o}_{0:T})$ and computing either the maximum-likelihood or *maximum a posteriori* assignment to the hidden variables. The learning problem in DBNs involves computing $P(\Theta|\mathcal{O})$ where Θ is a set of parameters, and $\mathcal{O} = \{\mathbf{o}^1_{0:T}, ..., \mathbf{o}^m_{0:T}\}$ is a set of m observation sequences. $P(\Theta|\mathcal{O})$, the maximum-likelihood or *maximum a posteriori* assignment of the parameters is computed. It is worth noting that the learning problem is really just an inference problem where the parameters are treated like random variables.

3 Generalized Evidence

First, let us consider some of the fundamental limitations of "traditional" evidence. An observation sequence, $\mathbf{o}_{1:T}$, describes a specific trajectory for a set of observed variables. We can then pose inference problems such as: *what is the probability distribution over the hidden variables given* $\mathbf{o}_{1:T}$, (i.e., $P(\mathbf{H}|\mathbf{o}_{1:T})$? We cannot, however, pose inference problems such as: *what is the probability distribution over the hidden variables given that the observed variables do not behave behave like* $\mathbf{o}_{1:T}$ (i.e., $P(\mathbf{H}|\neg\mathbf{o}_{1:T})$? Nor can we expressly prohibit certain behaviors in the hidden variables. For example, suppose we want to know about the probability distribution over hidden variables given $\mathbf{o}_{1:T}$ *and* where $h_i < c$, where c is an arbitrary constant. These represent just some of the limitations of traditional observation sequences. Our goal is to eliminate these limitations in the context of inference and learning. The fundamental problem is that a set of observation sequences defines a constant-sized set of behaviors, and cannot encode the notion of forbidden

behavior. We address this problem by introducing the notion of generalized evidence, wherein we specify allowed and forbidden behaviors using formulas in *temporal logics*.

3.1 Temporal Logic

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 [15]. There are a number of different temporal logics from which to chose, and different logics have different expressive powers. In this report, we consider *Linear temporal logic* (LTL), although many of these ideas can be extended to Computation Tree Logic (CTL), and other temporal logics. In particular, extensions to temporal logics specifically designed for probabilistic systems (e.g., PCTL) will be discussed in a future report. A complete discussion of LTL and temporal logics is beyond the scope of this paper. The interested reader is directed to [8] for more information.

Let $AP = \{a_1, ..., a_q\}$ be a finite set of *atomic propositions*, which are simply Boolean random variables that we introduce to answer specific questions about the state variables in the DBN. For example, if the sample space random variable X_i is $\{x_i^1, ..., x_i^k\}$, we might create an atomic proposition that evaluates to true iff X_i is some particular value (e.g., $a \Leftrightarrow X_i = x_i^j$). More generally, we can create atomic propositions that evaluate to true if a given variable is in a particular subset or range (e.g., $a \Leftrightarrow X_i \in \{x_i^1, x_i^7\}$, or $a \Leftrightarrow X_i \leq x_i^j$). Atomic propositions will be used later on to construct formulas describing both allowed and forbidden behaviors.

The syntax of an LTL formula, ϕ, is given by the following grammar:

$$\phi ::= a \mid true \mid (\neg\phi) \mid (\phi_1 \vee \phi_2) \mid (\phi_1 \wedge \phi_2) \mid (\phi_1 \rightarrow \phi_2) \mid (\phi_1 \Leftrightarrow \phi_2) \mid \bigcirc\phi \mid \mathbf{G}\phi \mid \mathbf{F}\phi \mid \phi_1 \mathbf{U} \phi_2 \mid \phi_1 \mathbf{R} \phi_2 \tag{1}$$

Here, $a \in AP$, is an atomic proposition; "true" is a Boolean constant; $\neg$, $\vee$,$\wedge$,$\rightarrow$, and $\Leftrightarrow$ are the normal logical operators; $\bigcirc$ is the "next" temporal operator indicating that ϕ must be true in the next state in the path[1]; **G** is the "global" temporal operator indicating that ϕ must be true on the entire path; **F** is the "future" temporal operator indicating that ϕ must eventually be true; **U** is the "until" temporal operator indicating that ϕ_1 holds until ϕ_2 holds; **R** is the "release" temporal operator indicating that ϕ_2 is true until the first position in which ϕ_1 is true, or forever if such a position does not exist. The temporal operator definitions, naturally, assume that each ϕ is a well-formed LTL formula. Additionally, given these, additional operators can be derived. For example, "false" can be derived from "$\neg true$", and exclusive disjunction ($\phi_1 \not\Leftrightarrow \phi_2$) is $\phi_1 \Leftrightarrow \neg\phi_2$

Notice that an LTL formula can describe either finite or infinite sequences. An LTL formula is satisfied by a sequence, $\pi_{0:T}$ if and only if it is satisfied for position π_0 on that path. The semantics of LTL formulas are defined recursively:

$\pi \models a$ iff a $\in \mathcal{L}(\pi[0])$

$\pi \models true, \forall\pi$

$\pi \models \neg\phi$ iff $\pi \not\models \phi$

[1]We note that the "next" operator is often written as $\mathbf{X}\phi$ in the literature. We use the alternative notation $\bigcirc$ to avoid confusion with the random variable **X**.

$\pi \models \phi_1 \vee \phi_2$ iff $\pi \models \phi_1$ or $\pi \models \phi_2$

$\pi \models \phi_1 \wedge \phi_2$ iff $\pi \models \phi_1$ and $\pi \models \phi_2$

$\pi \models \phi_1 \rightarrow \phi_2$ iff $\pi \models \neg\phi_1$ or $\pi \models \phi_2$

$\pi \models \phi_1 \Leftrightarrow \phi_2$ iff ($\pi \models \phi_1$ and $\pi \models \phi_2$) or ($\pi \models \neg\phi_1$ and $\pi \models \neg\phi_2$)

$\pi \models \bigcirc\phi$ iff $\pi[1] \models \phi$

$\pi \models \mathbf{G}\phi$ iff $\forall i \geq 0 \pi[i] \models \phi$

$\pi \models \mathbf{F}\phi$ iff $\exists i \geq 0, \pi[i] \models \phi$

$\pi \models \phi_1 \mathbf{U} \phi_2$ iff $\exists i \geq 0, \pi[i] \models \phi_2 \wedge \forall j < i, \pi[j] \models \phi_1$

$\pi \models \phi_1 \mathbf{R} \phi_2$ iff $\pi \models \neg(\neg\phi_1 \mathbf{U} \neg\phi_2)$

Here, the notation "$\pi \models \alpha$" means that π *satisfies* or *models* α. We note that there are a variety of useful variations on these operators, such as the bounded until (e.g., $\phi_1 \mathbf{U}^{\leq k} \phi_2$) which is true iff $\exists\, i : 0 \leq i \leq k, \pi[i] \models \phi_2 \wedge \forall j < i, \pi[j] \models \phi_1$.

Example 1: Our first example demonstrates that it is possible to encode a traditional observation sequence using an LTL formula. Suppose that $\mathbf{X} = \{X_1, X_2\}$, is a pair of Boolean random variables, that $\mathbf{o} = \{X_1\}$, and that $\mathbf{o}_{0:T} = (0, 1, ..., 0)$. We can define a set of atomic propositions $a_0, ..., a_T$ such that a_i is true iff $X_1(i) = \mathbf{o}(i)$. The LTL formula $\phi := a_0 \wedge a_1 \wedge ... \wedge a_T$ is equivalent to $\mathbf{o}$. Informally, the formula says that at time $t = 0$ $X_1 = 0$, at time $t = 1$ $X_1 = 1$, and so forth. This example could be easily extended to model an observation sequence that included more than one variable.

Example 2: Our second example illustrates a behavior that cannot be encoded using observation sequences. Let $\mathbf{h} = \{X_2\}$ be our hidden variable, and let a_1 and a_2 be a pair of atomic propositions that are true if $X_2(2) = 1$ and $X_2(10) = 0$, respectively. The LTL formula $\phi := \neg a_1 \wedge \neg a_2$ defines the set of all behaviors of the hidden variables such that a_1 and a_2 are false. This formula conceptually describes a negative observation sequence or a forbidden behavior.

Example 3: Our third example demonstrates a single formula that would require at least a quadratic number of observations sequences to encode. Consider an atomic proposition a that is true whenever X_1 is 1, regardless of the time index. Consider the LTL formula $\phi := \neg a \mathbf{U}^{\leq k} a$. This formula describes the set of observation sequences such that proposition a becomes true at or before time k. If $k = 2$, the formula is equivalent to the following set of observation sequences over X_1 : $\{(1), (0, 1), (0, 0, 1)\}$. Notice here that the formula also encodes observation sequences of different lengths. Traditional algorithms for inference and learning generally assume that the observation sequences are all the same length.

Example 4: Our final example demonstrates that a single formula can encode sequences of infinite length. Consider the LTL formula $\phi := \mathbf{G}a$. This formula describes the set of sequences (including sequences of infinite length) where proposition a is always true. Traditional algorithms for inference and learning assume that the observations sequences are finite length.

These examples demonstrate some of the possibilities of using LTL formulae for defining generalized evidence. The key observation is that a formula can encode a superset of the behaviors

that can be encoded using a finite set of observation sequences. Of course, it is not immediately obvious that it is actually possible to compute, say, $P(\Theta|\phi)$, for arbitrary ϕ. Fortunately, the field of Model Checking, which is reviewed in the next section, has a variety of algorithms for solving these problems.

4 Model Checking

The term *model checking* [8] refers to a family of techniques from the formal methods community for verifying properties of complex systems, such as digital circuits. 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 a variety of kinds of finite-state and hybrid (i.e., mixtures of finite and continuous variables) transition systems, including those with non-deterministic (i.e., asynchronous) and 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 human user.

There are three main elements of model checking. The *first* element is a formal specification of the system. In Section 4.1 we will introduce Kripke structures, which are a formal specification of concurrent systems. The *second* element is a formula in temporal logic, which was discussed in Section 3.1. The *third* element, naturally, is an algorithm for solving the model checking problem, which will be discussed in Section 4.3.

We will show in Section 4.4 that Kripke structures can encode DBNs. We will then show in Section 4.5 that DBNs can be used to symbolically encode Kripke structures. These observations are significant because it means that existing inference and learning algorithms for DBNs can be used to solve certain Model Checking problems; we are presently investigating these applications and will present them in a future publication. This report focuses on the other possibility; that is, using existing model checking algorithms for solving inference and learning problems in DBNs.

4.1 Kripke Structures

A *Kripke structure*, $\mathcal{M}$ is a tuple, $\mathcal{M} = (S, S_0, R, \mathcal{L})$. Here, S is a finite set of states, $S_0 \subseteq S$ is a set of starting states, $R \subseteq S \times S$ is a total transition relation between states, and $\mathcal{L} : S \mapsto 2^{AP}$ 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, Kripke structures for Markov chains can be constructed where 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).

Given a Kripke structure and a formula in temporal logic, ϕ, the *model checking problem* is to determine whether $\mathcal{M} \models \phi$. A complete discussion of model checking theory and practice is beyond the scope of this report. The interested reader is directed to [8] for a detailed treatment of the subject. We note, however, that model checking algorithms are exact. Additionally, when a

model does not satisfy the formula, model checking algorithms return a *counterexample*, which is a trace execution of $\mathcal{M}$. We will use these counterexamples to perform inference and learning.

Model checking algorithms are exact, which means that care must be taken if the state space of the model is very large. One of the most important contributions of the field of model checking is a set of powerful, general-purpose techniques for *symbolically* encoding and computing over Kripke structures. These symbolic methods, which are discussed further in Section 4.2, often allow model checking algorithms to verify properties many orders of magnitude larger than those that can be verified using an explicit search. We note that model checking algorithms are not a panacea; there are systems where even implicit methods do not work. In practice, however, model checking is often extremely successful, as evidenced by its industrial applications.

4.2 Symbolic Encodings of Kripke Structures

The phrase *symbolic model checking* refers to any technique whereby sets of states and the transitions between state are represented *implicitly* using Boolean functions. As a trivial example, consider a toy system with two binary variables, v_1 and v_2. If the set of allowable states for this system is $\{(11), (01), (10)\}$, we can efficiently encode this set of states using the following *characteristic function*: $v_1 \vee v_2$. Such Boolean formulas can be efficiently represented and manipulated in two ways: using Binary Decision Diagrams (BDDs), Multi-terminal Binary Decision Diagrams (MTBDDs), or using propositional satisfiability (SAT) formulae, which are discussed in the following sections. In practice, symbolic model checking algorithms based on SAT encodings generally scale to larger systems than those based on BDD encodings. This is primarily due to the efficiency and power of modern-day propositional SAT solvers[2]. Of course, SAT is *the* prototypical NP-hard problem, which implies that one cannot guarantee that SAT-based model checking algorithms will always succeed.

In practice, the construction of symbolic encodings of Kripke structures into BDDs, MTBDDs, or instances of SAT (see next section) is done automatically from a high-level language describing the finite-state system and its behavior. That is, it is generally not necessary to first construct the explicit state space. This is important, because the systems we will consider are too large to represent explicitly. In this report, we use the specification language used in the symbolic model checking tool NuSMV [5].

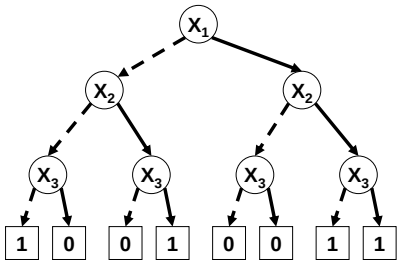
4.2.1 Binary Decision Diagram Encodings

One technique for symbolically encoding Kripke structures, which ultimately facilitated industrial applications of model checking, is the reduced ordered Binary Decision Diagrams (BDDs), introduced by Bryant [4] (Fig. 4). 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

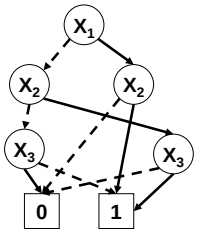
[2]It is worth noting that modern SAT solvers are capable of solving instances of SAT with hundreds of thousands of variables (or more), and millions of clauses [6].

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)

Figure 4: (**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.

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 [4], 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 the BDD representations of the functions in time proportional to $O(|f||g|)$, where $|f|$ is the size of the BDD encoding f. Model checking algorithms, which call APPLY and COMPOSE as subroutines, are then used to determine whether or not the system satisfies a given temporal logic formula.

Note that BDDs can be generalized to Multi-terminal BDDs [7] (MTBDD), which encode discrete, real-valued functions of the form $f : \{0,1\}^n \mapsto \{c_1, ..., c_k\} : c_i \in \mathbb{R}$. Significantly, MTBDDs can be used to encode real-valued vectors and matrices, and algorithms exist for performing matrix addition and multiplication over MTBDDs [7]. These algorithms play an important role in several model checking algorithms for stochastic systems [2]. MTBDDs can, in theory, be used to implement potential functions for Bayesian Networks.

4.2.2 Relationship Between BDDs and Bayesian Networks

There are some similarities between BDDs and Bayesian networks. Both data structures encode functions mapping a set of states to real numbers. They are both factored representations of functions and if the functions can be factored into a combination of much smaller functions, the space-savings can be dramatic. Indeed one might argue that the primary advantage of BDDs and graphical models is their compact nature. Unfortunately, there are some Boolean functions that cannot be efficiently encoded using BDDs, just as there are probability distributions that cannot be efficiently encoded as a Bayesian network (e.g., consider a Bayesian network where one node v_i has $n-1$ parents).

Evaluating functions in BDDs and graphical models is quite different, however. In a BDD function evaluation is performed by following a path from the root of the BDD to a leaf. In a Bayesian network, function evaluation is performed by computing sums and products over nodes.

Notice, however, MTBDDs can, in theory, be used to implement conditional probability functions for finite-state models.

4.2.3 SAT Encodings

The second way to symbolically encode Kripke structures is to reduce them to an instance of SAT. This is done in an iterative fashion by performing a kind of depth-first search looking for counterexamples. This procedure, which is known as *bounded model checking*, starts with $k = 1$ and iteratively increases k until the property is verified, or a counterexample is found. We note that even if the temporal logic formula concerns sequences of infinite length, it is possible to decide whether a given formula is true or false using bounded model checking. Informally, this is done by keeping track of states that loop back to some previously analyzed state.

The encoding of a Kripke structure as an instance of propositional satisfiability proceeds as follows: Given a model $\mathcal{M}$, a temporal logic formula, ϕ, and a bound k, it is possible to construct a propositional formula $\langle \mathcal{M}, \phi \rangle_k$ that will be satisfiable if and only if the property $\mathcal{M} \models^k \phi$. That is, if $\mathcal{M}$ satisfies ϕ for the first k steps (see [3] for a more detailed discussion).

The propositional formula, $\langle \mathcal{M} \rangle_k$, for an unrolled transition for a model $\mathcal{M}$ is defined as follows:

$$\langle \mathcal{M} \rangle_k := I(S_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$

where $I(S_0)$ is the characteristic function of the set of initial states, and $T(s_i, s_{i+1})$ is the characteristic function of the transition relation.

It is also possible to form a propositional formula $\langle \phi \rangle_k$ that is true if and only if the formula ϕ is valid along a path of length k. The conjunction of $\langle \mathcal{M} \rangle_k$ and $\langle \phi \rangle_k$ gives us the propositional formula we seek that will be true if and only if $\mathcal{M} \models^k \phi$. We will present a detailed example of a SAT-based encoding in Section 4.5.

4.3 Model Checking Algorithms

A model checking algorithm takes a Kripke structure, $\mathcal{M} = (S, R, \mathcal{L})$, and a temporal logic formula, ϕ, and finds the set of states $S^* \in S$ that satisfy ϕ: $\{s \in S \mid \mathcal{M}, s \models \phi\}$. The model checking problem is thus solved by determining whether $S^* \cap S_0 = \{\}$, or not. The complexity of model checking algorithms varies with the temporal logic and the operators used. For the types of formulas used in this report (see Sec. 6), an explicit state model checking algorithm requires time $O(|\phi|(|S| + |R|))$, where $|\phi|$ is the number of sub-formulas in ϕ [8]. Of course, for very large state spaces, even linear time is unacceptable. Symbolic model checking algorithms (see Section 4.2) operate on symbolic encodings (BDDs or SAT) of the Kripke structure and the formula. BDD-based symbolic model checking algorithms rely on the fact that the temporal operators of LTL 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 $\tau(S') = S'$. BDD-based symbolic model checking algorithms define an appropriate function, based on the formula, and then iteratively find the fixpoint of the

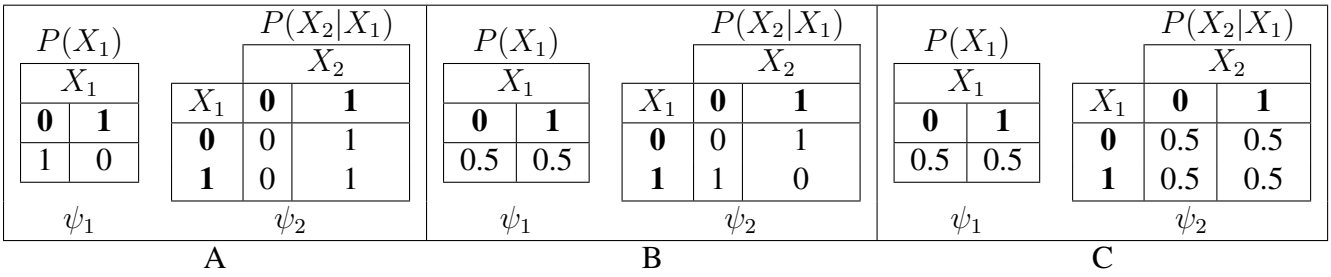


Table 1: Tabular representations of the potential functions for the Bayesian Network shown in Figure 1-A. The different functions encode different states for a hypothetical Kripke structure (i.e., S_0). (**A**) $S_0 = \{(X_1 = 0, X_2 = 1)$. (**B**) $S_0 = \{((X_1 = 0, X_2 = 1), (X_1 = 1, X_2 = 0)\}$. (**C**) $S_0 = \{(X_1 = 0, X_2 = 0), (X_1 = 0, X_2 = 1), (X_1 = 1, X_2 = 0), (X_1 = 1, X_2 = 1)\}$.

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 [8], ch. 6 for more details. SAT-based bounded model checking algorithms rely on SAT solvers to find a satisfying assignment for the propositional formula, if one exists.

4.4 From Kripke Structures to DBNs

It is easy to see that DBNs represent yet a third option for symbolically encoding Kripke structures. The state-space, S, corresponds to the sample space Ω. The set of starting states, S_0, can be specified using a Bayesian Network (Sec. 2.1). Traditionally, model checking does not consider probability distributions over states, so it is sufficient to simply define a uniform prior over all the states in S_0, when constructing the Bayesian Network. The transition relation can be defined using a 2-slice Bayesian Network (Sec. 2.2). Once again, if the Kripke structure does not define a probability distribution over state transitions, the 2-slice Bayesian Network can be defined such that there is a uniform probability over all allowable state transitions.

Example 5: Suppose that X_1 and X_2 in Figure 1-A are Boolean random variables, and that we want to specify the initial condition, $S_0 = \{(X_1 = 0, X_2 = 1)\}$. There are a number of equivalent ways of constructing graphical models encoding S_0. For example, we can use the Bayesian Network in Figure 1-A and define the functions shown in Table 1-A. We can also encode more complicated starting conditions. For example, suppose that we want to specify the initial condition, $S_0 = \{(X_1 = 0, X_2 = 1), (X_1 = 1, X_2 = 0)\}$. Once again, we can use the Bayes Net in Figure 1 and define the functions shown in Table 1-B. Naturally, if we want the starting conditions to include the entire sample space (i.e., $S_0 = \{(X_1 = 0, X_2 = 0), (X_1 = 0, X_2 = 1), (X_1 = 1, X_2 = 0), (X_1 = 1, X_2 = 1)\}$), we can also construct a suitable set of potential functions (Table 1-C).

Finally, the labeling function $\mathcal{L}$ over a set of atomic propositions can be encoded by creating a set of Boolean random variables, one for each atomic proposition at each time index $t = 0, 1, ...$ Conditional probability functions would be appropriately defined. For example, if atomic propo-

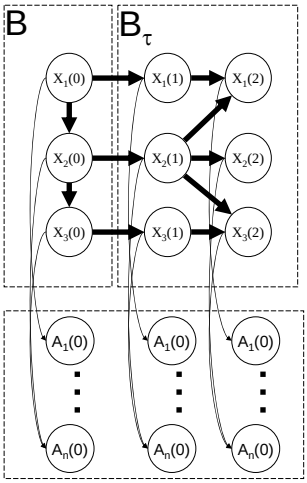


Figure 5: One possible encoding for a Kripke structure as a DBN. The nodes labeled B correspond to a prior distribution over the states, S_0. The nodes labeled B_τ correspond to the transition relation. The remaining nodes implement the labeling function, $\mathcal{L}$.

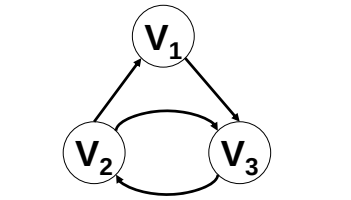
sition a_i is true if state variable X_i is less than some constant c, then the conditional probability function corresponding to the random variable implementing a_i would assign a uniform probability to any value of X_i that satisfies the proposition. Figure 5 shows the structure of one possible DBN encoding a Kripke structure. We note that the exact set of atomic propositions varies according to the formula being considered. Therefore, Figure 5 represents just one possible way to translate a Kripke structure into a DBN.

4.5 From DBNs to Kripke Structures

It is also possible to convert certain, restricted classes of DBNs into Kripke structures. The most restricted class of DBNs include Boolean Networks.

Example 6: Boolean Networks A Boolean Network 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 transfer functions that collectively define the discrete dynamics of the network. 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 $Pa(v_i) \subseteq V$ be the parents of v_i in the directed graph and let $k_i = |Pa(v_i) \cup \{v_i\}|$. A node can be its own parent if we add a self-edge. Each Boolean *transfer function* $\psi_i : \{0,1\}^{k_i} \mapsto \{0,1\}$ defines the discrete 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 discrete dynamics of the entire Boolean Networks. An example Boolean Networks is shown in Figure 6-A.

A Boolean network has deterministic dynamics, and thus it is easy to see how it can be mapped to a Kripke structure. The sample space of the Boolean network, Ω, is isomorphic to the set of states of a Kripke structure. Since a Boolean network doesn't define a set of starting states, we simply define $S_0 = S$. The Boolean transfer functions define a transition relation. A Boolean



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

(A)

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

(B)

Figure 6: (**A**) A Boolean Network. A Boolean Network 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. (**B**) A transition relation encoding the same dynamics as the Boolean Network. Notice that the Boolean Network is a compact encoding of the transition relation.

network does not define a set of atomic propositions. However, as shown in [10, 11], it is possible to perform model checking on a Boolean network. In particular, given a formula in temporal logic, we can define an appropriate set of atomic propositions, and the corresponding labeling function, $\mathcal{L}$.

In general, if **X** is a set of discrete random variables, then we can define an equivalent extended Kripke structure $(S, P(S), T, \mathcal{L})$, where $P(S)$ defines a probability distribution over S and T is a stochastic transition matrix. We note that model checking algorithms also exist for hybrid systems — mixtures of discrete and continuous valued variables. Thus, in principle, one can perform model checking on any DBN. In this report, however, we restrict the discussion to DBNs over finite-state models.

Example 7: We can encode the Boolean network in Fig. 6 as an instance of SAT (see Sec. 4.2.3). The characteristic function for an initial state, say $(\neg v_1 \wedge \neg v_2 \wedge \neg v_3)$, can be expressed using the formula $(\neg v_1^0 \wedge \neg v_2^0 \wedge \neg v_3^0)$, where v_i^t is the state of variable v_i at time index t. The characteristic function describing the first step is: $T(S_0, S_1) = ((v_1^1 \leftrightarrow (v_1^0 \wedge v_2^0)) \wedge (v_2^1 \leftrightarrow \neg v_3^0) \wedge (v_3^1 \leftrightarrow (v_1^0 \wedge v_2^0)))$, where $\leftrightarrow$ represents the logical XNOR. Similarly, the characteristic function describing the second step is: $T(S_1, S_2) = ((v_1^2 \leftrightarrow (v_1^1 \wedge v_2^1)) \wedge (v_2^2 \leftrightarrow \neg v_3^1) \wedge (v_3^2 \leftrightarrow (v_1^1 \wedge v_2^1)))$.

Taking the conjunction of these characteristics we obtain:

$$\begin{aligned}\langle \mathcal{M} \rangle_2 \quad := \quad & (\neg v_1^0 \wedge \neg v_2^0 \wedge \neg v_3^0) \wedge ((v_1^1 \leftrightarrow (v_1^0 \wedge v_2^0)) \wedge (v_2^1 \leftrightarrow \neg v_3^0) \wedge (v_3^1 \leftrightarrow (v_1^0 \wedge v_2^0))) \\ & \wedge ((v_1^2 \leftrightarrow (v_1^1 \wedge v_2^1)) \wedge (v_2^2 \leftrightarrow \neg v_3^1) \wedge (v_3^2 \leftrightarrow (v_1^1 \wedge v_2^1))).\end{aligned}$$

Clearly, this can be extended for as many steps as desired. We can then call a propositional SAT solver to find a satisfying assignment to this formula. Moreover, we could construct an instance of

SAT that is true iff a given LTL formula, ϕ, is true. For example, let $\phi = v_1^2 \wedge v_2^2 \wedge v_3^2$, which is true if $X_1 = 1$,$X_2 = 1$, and $X_3 = 1$ at time $t = 2$. Taking the conjunction of the characteristic functions describing the dynamics and ϕ we obtain:

$$\begin{aligned}\langle \mathcal{M}, \phi \rangle_2 \quad := \quad & (\neg v_1^0 \wedge \neg v_2^0 \wedge \neg v_3^0) \wedge ((v_1^1 \leftrightarrow (v_1^0 \wedge v_2^0)) \wedge (v_2^1 \leftrightarrow \neg v_3^0) \wedge (v_3^1 \leftrightarrow (v_1^0 \wedge v_2^0))) \\ & \wedge ((v_1^2 \leftrightarrow (v_1^1 \wedge v_2^1)) \wedge (v_2^2 \leftrightarrow \neg v_3^1) \wedge (v_3^2 \leftrightarrow (v_1^1 \wedge v_2^1))) \wedge v_1^2 \wedge v_2^2 \wedge v_3^2.\end{aligned}$$

This formula cannot be satisfied, which is correct, since the property does not hold for the Boolean Network in Fig. 6. The fact that this formula cannot be satisfied indicates that the probability $P(\mathbf{X}|\neg v_1^0 \wedge \neg v_2^0 \wedge \neg v_3^0 \wedge \phi)$ is zero, for all possible assignments of $\mathbf{X}$.

5 Inference and Learning in DBNs over Generalized Evidence

In Section 2.4 we defined the traditional inference and learning problems in DBNs. We then introduced the notion of generalized evidence in Section 3. The previous section introduced model checking, and demonstrated that a restricted class of DBNs can be encoded as Kripke structures. This, in turn, implies that: a) this class of DBNs can be symbolically encoded using BDDs, MTBDDs, or as instances of SAT; and b) one can perform model checking on DBNs. Taken together, all of this implies that we can solve inference and learning problems in DBNs using symbolic model checking algorithms. We first demonstrated this in [10, 11].

5.1 Algorithm

The algorithm for solving the inference and learning problem over generalized evidence involves three steps:

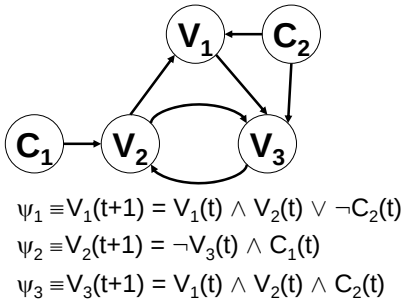
1. The DBN must be translated into a high-level language for specifying Kripke structures. Figure 8 gives an example for the Boolean network in Figure 7. More generally, the high-level encoding will define state variables, the dynamics of the model, including the parameters (i.e., probabilities).

2. A temporal logic formula must be written that encodes the generalized evidence.

3. A suitable symbolic model checking algorithm to find a satisfying assignment for $\mathbf{h}$ or Θ.

Example 8: Inference We extend the model in Fig. 6-A to add a pair of 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. 7). Control nodes have no parents and represent externally manipulatable variables.

Consider a set of initial states, $\mathfrak{I}$, for the nodes in V specified in terms of a characteristic Boolean function. For example, the expression $\mathfrak{I} = (v_1 \wedge \neg v_2 \wedge v_3)$ defines the set $\{(1, 0, 1)\}$, and $\mathfrak{I} = (v_1 \wedge v_3)$ defines the set $\{(1, 0, 1), (1, 1, 1)\}$. We define a set of goal states, $\mathfrak{F}$, in a similar



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

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

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

(A)

	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		

(B)

Figure 7: (**A**) A Boolean Network with two control nodes (C_1 and C_2). (**B-top**) An initial state and time-sensitive goal. (**B-bottom**) A control policy (last two columns) that achieves the goal at the specified time.

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 Boolean network control problem is to find a control policy that drives the Boolean network such that $\mathbf{v}(0) = \mathfrak{I}$ and $\mathbf{v}(t) = \mathfrak{F}$. This is a generalized inference problem that we will solve using model checking algorithms. Specifically, our goal is to algorithmically generate control policy, Γ, for a given Boolean network, B, initial set of state, $\mathfrak{I}$, a set of goal states, $\mathfrak{F}$, and end time, t, — or to guarantee that no such policy exists.

The first step in solving this problem is to compile the Boolean network into a form suitable for model checking. Figure 8, shows pseudo-code for encoding the Boolean Network in figure 7. 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.

The second step in solving this problem is construct a suitable temporal logic formula. Our goal is to create a formula that is true if it is possible to end up in the goal state(s), $\mathfrak{F}$, at time t. Let ϕ_F be a formula describing the goal state. Let $\phi_t :=$COUNTER$= t$ be a Boolean formula that evaluates to true if the variable COUNTER is t. The formula $\phi := \neg\phi_F \mathbf{U}(\phi_F \wedge \phi_t)$ can be used to find a control policy. This formula is true in a path that first enters state $\mathfrak{F}$ at time t. Alternatively, if we wish to relax the restriction that the Boolean network cannot enter state $\mathfrak{F}$ before time t, we would use the formula $\phi := true\mathbf{U}^{\leq t}\phi_F$. This formula is true in a path that first enters state $\mathfrak{F}$ at or before time t. Temporal logics are very expressive and can encode a number of complex behaviors. For example, it is possible to specify particular milestones through which the model should pass en route to the final goal. That is, one can construct formula that say that the Boolean Network 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 generalized approach to inference and learning.

The final step in solving this problem is to call an appropriate symbolic model checking algorithm to find a control policy. If a control policy exists (i.e., if ϕ is true), we want a witness to the formula because it will reveal the states of the control nodes which are, of course, a valid control policy. Model checking algorithm, however, usually don't report witnesses, but counterexamples. Thus, we can ask for a counterexample to $\neg\phi$ to obtain the control policy.

Example 9: Learning Suppose that we know the topology of a Boolean network, but we don't know the Boolean transfer functions (i.e., each ψ_i). If we have a set of observation sequences $\mathbf{o}_{0:T}$ it is possible to *learn* the Boolean transfer functions. For example, [] showed that it is possible to learn a Boolean network given $O(\log n)$ randomly sampled input-output pairs. This result assumes that the input-output pairs are uniformly distributed over the sample space of the model.

In some domains, it may be reasonable to assume that it is possible to obtain a uniformly sampled set of input-output pairs, but in other domains it is not possible. In Biology, for example, it is generally very difficult, or impossible to set up an experiment in an arbitrary starting state. Additionally, a Boolean network is a discrete-time model, which means that it necessary to select a ΔT corresponding to the Boolean transfer functions. In a Biological experiment it is generally necessary to take measurements from multiple cells. Synchronizing cells can be very difficult and it is therefore difficult to obtaining a set of input-output pairs that have been sampled at exactly ΔT intervals. Moreover, because Biological measurements are subject to noise, obtaining a self-consistent set of input-output pairs may simply be impossible.

In Biology it is far more likely that the available data correspond to steady-state behaviors that are observed when the system is started from some starting state. We can encode these behaviors as formulas in temporal logic. For example, if ϕ_a is a formula that is true when the system is in the starting state, and ϕ_b is a formula that is true when the system is in the steady state, then the formula $\phi := \phi_a \rightarrow G\phi_b$ is true if the state ϕ_a ends in the steady-state behavior ϕ_b. This is an example of a generalized learning problem which we can solve using model checking.

Like the previous example, we can encode the Boolean network in a high-level language. The primary difference is that we need to add random variables that collectively define the Boolean transfer functions. There are 2^k rows in the truth table for a random variable with k parents in the graph. We will therefore create 2^k random variables for each ψ_i. We note that in a Biological network, k is likely to be small. An assignment to these random variables defines a Boolean transfer function. The generalized learning problem involves finding an assignment that makes a given formula ϕ true. We note that in general, there may be more than one assignment that satisfies a given formula. If available, prior knowledge about the nature of the transfer function can, of course, be encoded into the network by restricting the space of possible transfer functions.

There may be more than one than one known steady-state for the model, corresponding to different starting states. Each of these (starting state, steady-state) pairs can be encoded as a separate formula in temporal logic. We can then construct a master formula, ϕ, by taking the union of these formulas. We can therefore solve the generalized learning problem by calling a model checking algorithm to find a counterexample to $\neg\phi$.

In the previous two examples, the DBNs were Boolean Networks, which have deterministic dynamics. If the DBN is a model of a finite-state stochastic process, we can easily identify assignments with non-zero probabilities through counterexample generation. If we extend our set

```
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 ;
```

Figure 8: Pseudocode based on the language used in the symbolic model checking program NuSMV. This code implements the Boolean Network in Figure 7. 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.

of atomic proposition to keep track of the joint probability of a given assignment, we can also perform a binary search over these probabilities to find maximum-likelihood assignments. We will demonstrate this in Section 6. We note, however, that our approach is limited in its scope and that more research is needed to develop techniques for solving the general problem.

6 Case Studies

In this report we are only considering finite-state models. Moreover, we assume that each potential function is a finite function. That is, each function has the form: $\psi_i : \mathcal{X}_{N(i)} \times \mathcal{X}_i \rightarrow \{c_1, ..., c_z\} : c_i \in \mathbb{R}$. For example, consider the case where each variable is a Boolean random variable, and each potential function is a Boolean function. In this case, each potential function has the $\psi_i : \{0,1\}^{|N(v_i)|} \rightarrow \{0,1\}$ and can be thought of as a truth table. If we treat each function as a random variable, then the state space of the variable corresponding to ψ_i is doubly exponential in the number of neighbors of variable v_i. That is, if variable v_i has $k = |N(v_i)|$ neighbors, then the truth table for ψ_i has 2^k elements, which implies that there are 2^{2^k} possible truth tables for ψ_i. Naturally, we are not limited to Boolean functions or Boolean random variables.

This section presents the results of a number of case studies on different kinds of Dynamic Bayesian Networks.

6.1 Inference in Boolean Networks with Synchronous Dynamics

We have previously reported the results of experiments on inference in Boolean networks with synchronous dynamics [9, 10, 11]. In that study, we performed a systematic study on more than

13,400 separate Boolean networks with synchronous dynamics. Among the models we considered was a Boolean network of *D. Melanogaster* embryogenesis [1]. The size of the networks ranged from 12 variables to 15,360 Boolean variables, and a variety of topologies were considered. The mean and median runtimes were 2 and 0.6 seconds, respectively. The longest runtime (693 seconds) was on a model with 84 variables, an average in-degree of 4, and a formula that specified a behavior consisting of 32 steps. The runtime on the largest model (15,360 variables), was 6.2 minutes. That model was a variation on the *D. Melanogaster* model.

In addition to these earlier studies, we have applied our method to a model of guard cell abscisic acid signaling [12]. The reader is directed to [12] for more information on the biological significance of this model. Here, we will focus on several variations on this model in this, and the following sections. The model has 43 Boolean random variables and describes the regulatory process controlling stomatal pores in plants (Fig. 9). These pores open and close to regulate the amount of water and carbon dioxide in the plant.

This model has two steady state behaviors. The first is where the stomatal pore (denoted by variable "closure" in Fig. 9) is closed which is characterized by variables 'ABA', 'CLOSURE', 'OST', 'SPHK', 'PH', 'PLD', 'ROP2', 'AGB' 'RCN', 'NIA12', 'InsPK','IP6', 'S1P', 'PA', 'ATRBOH', 'GPA', 'ROS', 'AnionEM', 'DEPOLAR', 'KOUT', and 'Actin', being in the on state and the variables 'CAIM', 'RAC', 'MALATE', 'ABI', and 'HATPase' are in the off state ; the remaining variables do not have a fixed value in the steady state. The second state is where the stomatal pore is open, which is characterized by variables 'AGB', 'RAC', are in the on state and the variables 'ABA', 'CIS', 'OST', 'SPHK', 'PH', 'RCN', 'NIA12', 'PLC', 'InsPK', 'IP6', 'ADPRc', 'GC', 'S1P', 'IP3', 'cADPR', 'cGMP', 'ATRBOH', 'ROS', 'ABI', and 'NO' are in the off state; the remaining variables do not have a fixed value in the steady state. Thus, we can define two formulas describing the steady states:

- $\phi_{closed} := ABA = 1 \wedge ... \wedge Actin = 1 \wedge CAIM = 0 \wedge ... \wedge HATPase = 0$
- $\phi_{open} := PEPC = 1 \wedge .... \wedge RAC = 1 \wedge ABA = 0 \wedge ... \wedge NO = 0$

The inference problem we will consider is showing that there exists a state where it is possible to reach ϕ_{closed} and remain there, and that there exists a state where it is possible to reach ϕ_{open} and remain there. Consider the following two LTL formulae: $\phi_1 =: \neg\phi_{closed} \rightarrow !FG\phi_{closed}$ and $\phi_2 =: \neg\phi_{open} \rightarrow !FG\phi_{open}$. The first formula says that if you are in a state where ϕ_{closed} is false, you can never reach a state where ϕ_{closed} is globally true. The second formula says that if you are in a state where ϕ_{open} is false, you can never reach a state where ϕ_{open} is globally true. Counterexamples for these formulas will reveal an assignment such that ϕ_{closed} (resp. ϕ_{open}) is false in the starting state but becomes globally true, which is exactly what we want.

The ABA signaling model was encoded in the NuSMV modeling language. The model consists of 43 Boolean variables updated in a synchronous fashion. The sample space therefore has $2^{43} \approx 10^{13}$ states. We then used SAT-based bounded model checking to find counterexamples for ϕ_1 and ϕ_2. These experiments took 0.03 and 0.19 seconds, respectively.

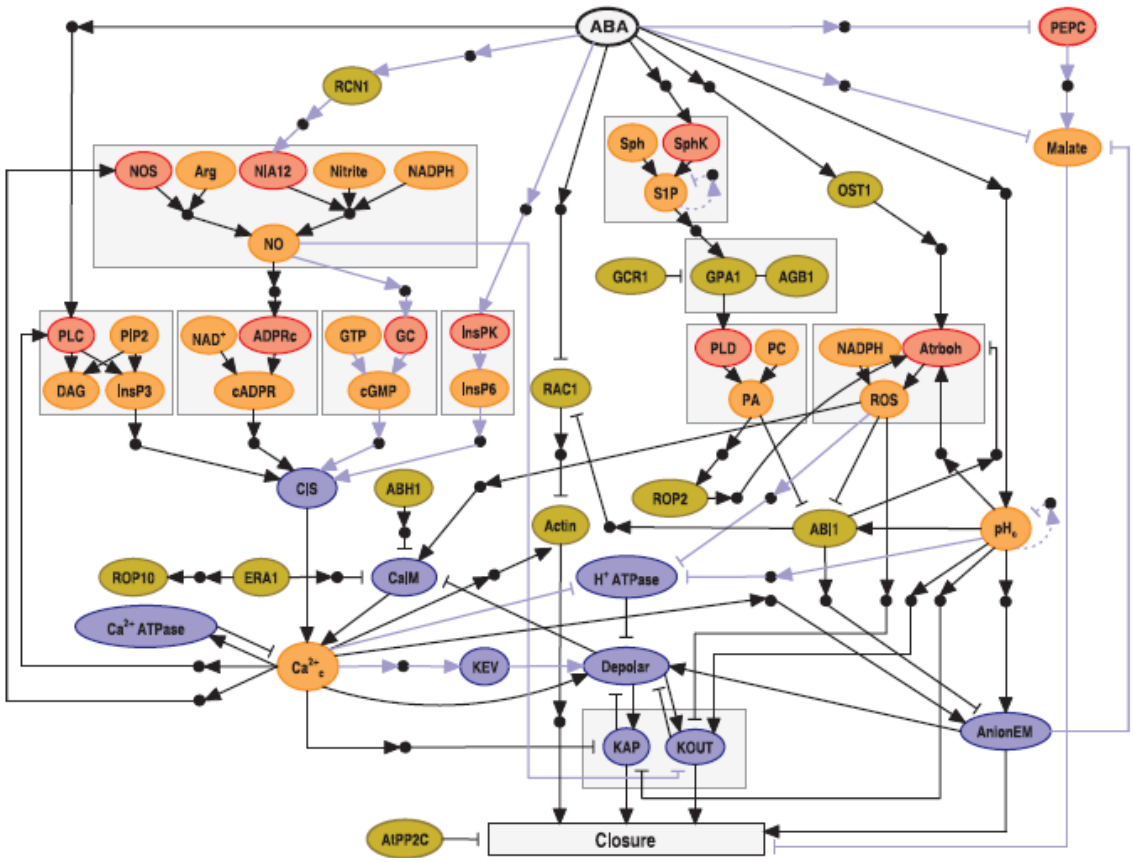


Figure 9: A Boolean Network model of Guard Cell ABA Signaling, as presented in [12]. Colors of the nodes represent the biological function of the component: enzymes are red, signal transduction proteins are green, membrane proteins are blue, and small molecules are orange. Small black nodes are putative intermediates. See [12] for more details. An arrow from node i to node j with a pointed tip indicates that node i promotes node j. An arrow from node i to node j with a flat tip indicates that node i inhibits node j.

6.2 Inference in Boolean Networks with Asynchronous Dynamics

We modified the model in the previous section so that it has asynchronous dynamics. There are several ways to implement asynchronous behaviors. The simplest way is to re-code the model to consist of n asynchronous processes, each modeling the behavior of one variable. We considered a different approach to modeling asynchronous behavior; we augmented the model in the previous section with a new variable, 'VSEL', which acts as a variable selector. 'VSEL' takes on values from 1 to 43. When 'VSEL' is 2, for example, only variable two is allowed to update. We chose this approach to demonstrate the scalability of the model. The resulting model had 43 Boolean variables and 1 integer-valued variable. The sample space therefore has $43 \times 2^{43} \approx 10^{14.6}$ states. SAT-based bounded model checking was used to find counterexamples for ϕ_1 and ϕ_2. These experiments both took 0.04 seconds to complete.

6.3 Learning in Boolean Networks with Synchronous Dynamics

We have previously reported the results of experiments on learning in Boolean networks with synchronous dynamics [9, 11]. In those studies, we inferred a set of Boolean transfer functions in the previously mentioned model of *D. Melanogaster* embryogenesis. The sample space for this learning problem consisted of 6.9×10^{10} possible Boolean network models. Our algorithm found a solution in 5.3 seconds using bounded symbolic model checking.

We have also used our method to learn the Boolean transfer functions for the synchronous version of the ABA signaling model. The learning problem for this network is to find a set of potential functions (Ψ) that will allow the Boolean network to satisfy ϕ_1 *and* ϕ_2. One way to do this is to declare random variables that collectively implement the Boolean transfer functions. For example, if variable v_1 has two parents, then the Boolean transfer functions corresponds to a truth table with 4 rows, one for each of the possible states of the variable's parents. There are therefore 4 truth assignments, which can be modeled using 4 Boolean variables. In the case of the ABA signalling model, the Boolean transfer functions can be implemented using 242 Boolean random variables.

The formula needed to ensure that the model learns to satisfy ϕ_1 and ϕ_2 at the same time is: $\phi := (!\phi_1 \wedge !\phi_2) - > !FG(\phi_1 \wedge \phi_2)$. Notice that if ϕ_1 and ϕ_2 are mutually incompatible, then there is no assignment that can satisfy ϕ. However, we can solve the learning problem by creating a model that, essentially, runs two copies of the ABA signalling model in parallel. This is done by duplicating the 43 state variables. The two sets of variables run independently, in the sense that they are allowed to start in different states. The dynamics of the model, however, are controlled by the same set of Boolean transfer functions. The learning problem is to find a set of Boolean transfer functions such that one copy of the ABA signalling model satisfies ϕ_1 and the other satisfies ϕ_2. The resulting model had $2 \times 43 + 242 = 328$ Boolean variables, corresponding to a sample space of $2^{328} \approx 10^{98.7}$ states. Nevertheless, our algorithm found a solution in 0.927 seconds using bounded symbolic model checking.

6.4 Learning in Boolean Networks with Asynchronous Dynamics

We modified the model in the previous section so that it has asynchronous dynamics. The resulting model had a sample space of $43 \times 2^{328} \approx 10^{100.4}$ states. SAT-based bounded model checking was used to find an assignment to the Boolean transfer function in 0.924 seconds.

6.5 Inference in Stochastic Models

We extended the ABA signaling model to create a simple stochastic model. A Boolean network's dynamics are deterministic; that is, the probability of each transition is exactly 1.0. However, we can establish a prior over starting states. The probability of any trace, therefore, is simply the prior probability.

We extended the model by adding one random variable called 'PRIOR'. That variable is initialized based on the number of 1s in the starting state. The number of 1's was mapped to a probability value. There are, therefore, 44 unique prior probabilities (0...43). That is, the sample

space of 'PRIOR' has size 44. Once initialized, 'PRIOR' does not change. It is therefore a proxy for computing the probability of the assignment. It would be straight-forward to extend this to consider more complicated priors or to keep track of the transition probabilities in truly stochastic models — provided that the transition probabilities can be encoded using tables.

We then extended the LTL formulas to include the prior. The new formulas are: $\phi_1 =: \neg\phi_{closed} \rightarrow !FG(\phi_{closed} \wedge PRIOR > p)$ and $\phi_2 =: \neg\phi_{open} \rightarrow !FG(\phi_{open} \wedge PRIOR > p)$, where p is a probability threshold. These formulas are true if, as before, it is possible to reach ϕ_{closed} and ϕ_{open} while ensuring that the prior probability on the initial state is greater than a given constant, p. It is possible to find a maximum-probability assignment by performing a binary search over p to find the largest p for which a satisfying exists.

The new model had $44 \times 2^{43} \approx 10^{14.6}$ possible states. SAT-based bounded model checking to find counterexamples for ϕ_1 and ϕ_2 in 0.95 and 0.684 seconds, respectively, once the maximum p had been identified. In searching for the maximum p to satisfy ϕ_1, the longest computation required 66.28 seconds. The longest computation required to find the maximum p to satisfy ϕ_2 also required less than 1 second. This behavior demonstrate that the runtime can vary, based on the details of the computation being performed. This is not unexpected since it is well-known that the efficiency of SAT-solvers is sensitive to the problem instance [13].

6.6 Inference in Asynchronous Stochastic Models

We modified the model in the previous section so that it has asynchronous dynamics. The resulting model had $43 \times 44 \times 2^{43} \approx 10^{16.2}$ states. SAT-based bounded model checking was used to find counterexamples for ϕ_1 and ϕ_2. These experiments took 195.986 and 1.914 seconds, respectively, once the maximum p had been identified. When searching for the maximum p values for ϕ_1 and ϕ_2, the longest runtimes were 24 minutes, and 2.513 seconds, respectively.

6.7 Learning in Stochastic Models

We defined a set of priors for the Boolean transfer functions in the ABA signalling model. The variable 'PRIOR' had 44 possible values. meaning that the state space had states. The resulting model had a sample space of $44 \times 2^{328} \approx 10^{100.4}$ states. SAT-based bounded model checking was used to find an assignment to the Boolean transfer function in 1.022 seconds.

6.8 Learning in Asynchronous Stochastic Models

Finally, we modified the model in the previous section so that it has asynchronous dynamics. The resulting model had a sample space of size $43 \times 44 \times 2^{328} \approx 10^{102}$ states. SAT-based bounded model checking was used to find an assignment to the Boolean transfer function in 1.028 seconds.

7 Conclusion and Future Work

In this report we have introduced the inference and learning problems in Dynamic Bayesian Networks over generalized evidence. Temporal logics are a powerful means for compactly encoding dynamic behaviors that are either impossible or inefficient to encode using a finite set of observation sequences. The case studies considered in this proposal were limited to properties expressed in LTL. The use of alternative logics, especially those incorporating probabilistic operators, is an interesting area for future research.

We have also discussed the relationship between Dynamic Bayesian networks and Kripke structures. Dynamic Bayesian Networks can be seen as a new symbolic encoding for Kripke structures. Like BDD and SAT based symbolic encodings, DBNs efficiently encode complex functions. The DBNs considered in this report are discrete-time models. We note, however, that there are DBNs describing continuous-time models. Such models may prove useful in the development of new model-checking techniques for continuous-time systems.

Finally, we have demonstrated that existing model checking algorithms can solve the inference and learning problem over generalized evidence, exactly, in theory and in practice, for a restricted class of DBNs. Our results are limited, however, to DBNs for finite-state systems with dynamics that can be encoded using finite-functions. DBNs themselves, however, are not limited to such models. DBNs can encoded hybrid systems, for example. An important area of future research will be to determine whether existing algorithms for model checking hybrid systems can be used for solving the inference and learning problem over generalized evidence in hybrid systems. Here, it is likely that Satisfiability Modulo Theory (SMT) solvers, and Iterative Abstraction Refinement can be used to address at least some of these problems in DBNs with linear dynamics. The development of algorithms for non-linear systems remains an important goal for future work.

This report has focused on the use of model checking algorithms for solving problems in DBNs. Naturally, it may be the case that algorithms for DBNs might be used to develop new algorithms for model checking probabilistic systems. The machine learning community has developed a variety of algorithms for solving the (traditional) inference and learning problems in graphical models. Significantly, some of these algorithms are capable of providing rigorous upper and lower bounds. Investigating the use of these algorithms in the context of model checking is also an interesting area for future 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] 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] 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.

[3] 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.

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

[5] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 359–364, London, UK, 2002. Springer-Verlag.

[6] 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.

[7] 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.

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

[9] C.J. Langmead and S.K. Jha. Symbolic approaches for finding control strategies in boolean networks. Technical Report CMU-CS-07-155R, Carnegie Mellon University, 2007.

[10] C.J. Langmead and S.K. Jha. Symbolic approaches to finding control strategies in boolean networks. *Proc. 6th Asia-Pacific Bioinf. Conf. (APBC)*, pages 307–319, 2008.

[11] C.J. Langmead and S.K. Jha. Symbolic approaches to finding control strategies in boolean networks. *J. Bioinf. and Comp. Biol.*, page under review, 2008.

[12] S. Li, S.M. Assmann, and R. Albert. Predicting Essential Components of Signal Transduction Networks: A Dynamic Model of Guard Cell Abscisic Acid Signaling. *PLoS Biology*, 4(10):1732–1748, 2006.

[13] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400(8):133–137, 1999.

[14] Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, 2002.

[15] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE. Foundations of Computer Science (FOCS)*, pages 46–57, 1977.