THE INSTITUTE FOR SYSTEMS RESEARCH

ISR TECHNICAL REPORT 2012-12

Compositional Analysis of Dynamic Bayesian Networks and Applications to CPS

Shahan Yang, Yuchen Zhou and John Baras

The Institute for Systems Research

A. JAMES CLARK SCHOOL OF ENGINEERING

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the A. James Clark School of Engineering. It is a graduated National Science Foundation Engineering Research Center.

www.isr.umd.edu

Compositional Analysis of Dynamic Bayesian Networks and Applications to CPS

Shahan Yang, Yuchen Zhou and John Baras

Abstract

Dynamic Bayesian networks (DBNs) can be effectively used to model various problems in complex dynamic systems. We perform an empirical investigation on compositional analysis of DBNs using abstraction. In static systems and hidden Markov models, computation of a metric called *treewidth* induces a tree decomposition that can be used to perform logical or probabilistic inference and $\max+$ optimizations in time exponential in treewidth and *linear* in overall system size. Intuitively, the linear scaling means that very large systems can be analyzed as long as they are sufficiently sparse and well structured. In these simple cases, summary propagation, which uses two operations, summation (projection) and product (composition), suffices to perform the inference or optimization. Here, we begin an extension of this to structured networks of communicating dynamic systems.

We define generalizations of projection and composition operators that treat labeled Markov chains as primitive objects. The projection operation, corresponding to summation, is implemented as label deletion followed by exact state reduction for Markov chains, similar to Hopcroft's DFA minimization algorithm, with $\mathbb{O}(n \log m)$ complexity. The composition operation is the product of state machines. We use canonical MDDs, similar to BDDs, to capture logical dependencies symbolically. Combining symbolic representations with Markov chain lumping algorithms is a novel contribution. Using this approach, we have created a tool leveraging model based systems engineering technologies. The communicating Markov chains are specified using UML Statecharts via Papyrus extended using an ANTLR parsed domain specific language (DSL).

The tool reduces the number of states in networks of Markov chains by several orders of magnitude. In one example, a network having a product state space of more than 600 million states is reduced to about 500 states. A feature of this technique is that the state space is examined incrementally, meaning that the full state space is never explicitly represented, even as an input to the reduction algorithm. The primary reduction appears to come from symmetry which is surprising because the technique includes no explicit symmetry handling. We note that composition is efficient at least for systems with high symmetry. We describe applications to a hospital intensive care unit (ICU) from a systems engineering perspective.

I. Introduction

A fundamental problem in systems engineering is decomposing large systems into smaller, more manageable pieces. Since a system has multiple behaviorally equivalent logical decompositions, for example, refactoring transformations [1] have been used by the software community to change the structure of object-oriented programs without affecting behavior, an inevitable question that arises is how to compare different decompositions, or more fundamentally, what is gained from these decompositions. Is it merely aesthetic or reflective of the physical configuration of the system or is there something more meaningful? For a large class of problems, as reviewed in [2], analysis complexity is tightly linked to the specific decomposition. The *treewidth* of a system, which is a metric based on its graphical decomposition, has a significant influence on analysis complexity. For many NP complete problems, analysis[1] is exponential in treewidth and linear in system size [3]. The linear scaling in problem size gives us hope that very large systems can be effectively analyzed if properly structured, avoiding the curse of dimensionality.

The question we begin to address here is whether this can be generalized to dynamic systems. Since these systems are physical in nature, it is reasonable to assume sparsity and structure in the communication graph, implying systems with low treewidth. However, it is commonly believed that because every variable becomes coupled over time, it is not possible to perform exact inference in complexity less than $\mathbb{O}(|Q|^N)$ where $|Q|$ is the number of states in each machine and N is the number of machines [4] (see Section II-A for details). In the more complexity oriented work of Ferrara [5], the author proves that nondeterministic automata networks having bounded *local treewidth*[2] are EXPSPACE complete, meaning we should not expect reductions for every system. While this is true in general (see Section VII for more details), our results demonstrate, by example, cases for which reduction is efficient.

Figure 1 shows a conceptual example of a dynamic Bayesian network. The high level structure is a simple tree, but each node of the tree contains an internal topology representing dynamic behavior. The semantics of composing the machines (see Section II-B) means that the ground topology once we reduce it to a single machine has a state space given by the Cartesian product of the sets of states of its constituent machines. It is very easy to describe machines with an immense number of states due to this exponential combination. However, this explosion under composition also appears in the simpler problems referred to earlier and in that context is avoided by summary propagation. Assuming the treewidth is not too large, it is sufficient to analyze a component at a time, taking local products and projecting out unnecessary intermediate information. In this sense,

S. Yang, Y. Zhou and J. Baras are with the Institute for Systems Research, University of Maryland, College Park. 2247 AV Williams Building, College Park, MD 20742 USA. Fax: 301-314-8486. Phone: 301-405-6606. E-mail: {syang, yzh89, baras}@umd.edu.

[1]Analysis can be logical inference, probabilistic inference, dynamic programming using $\max+$ algebra, etc.

[2]Local treewidth is the treewidth of the communication graph, as defined in [5]. Global treewidth in their terminology refers to the treewidth of the flattened graph.

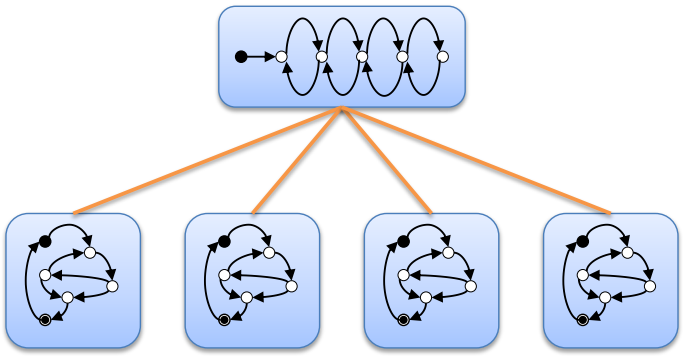


Fig. 1. A network of communicating machines is depicted above. Machines may exchange information if they are linked. There is a local and implicit global topology above. In staying consistent with [5], when we refer to the *local* topology, we mean the 5 rounded rectangles as nodes and the four links connecting them. The global topology is the single machine that results after flattening the above network using composition.

summary propagation can be seen as a methodical approach to analyzing a system using mathematically defined abstraction and composition operations. Intuitively, this work can be seen as an extension of summary propagation to more dynamic systems.

A. Our Contribution

- We present an encapsulation based approach to analyzing dynamic Bayesian networks.
- We present a novel algorithm that allows state reduction on symbolic Markov chains.
- We create a tool integrating UML Statecharts with a DSL extension for describing DBNs and a computational engine implementing the algorithms presented.
- We analyze a model of an intensive care unit with costs.

II. Labeled Dynamic Bayesian Network (DBN) Formalism

A labeled DBN consists of a number of communicating Markov chains where the transition matrix may depend on some output messages from a neighboring machine (for an excellent exposition on DBNs, see [6]). Unless otherwise specified, assume that every set defined is finite.

Definition 1. *Formally, we define a **DBN** as the tuple*

$$\Pi = \langle \mathcal{L}, P_1, \ldots, P_N \rangle$$

where

$$\begin{aligned} \mathcal{L} = \{\Sigma_1, \ldots, \Sigma_M\} \quad & \textit{is a set of system labels} \\ \textit{where each } \Sigma \in \mathcal{L} \quad & \textit{is a set of symbols and} \\ \textit{each } P_i \textit{ for } i = 1 \ldots N \quad & \textit{is a tuple as defined below.} \end{aligned}$$

$$P_i = \langle Q_i, \mathcal{I}_i, \mathcal{O}_i, \delta_i, \lambda_i \rangle$$

with components given in the following table[3].

$$\begin{aligned} Q_i \quad & \textit{set of states} \\ \mathcal{I}_i \subseteq \mathcal{L} \quad & \textit{input labels} \\ \mathcal{O}_i \subseteq \mathcal{L} \quad & \textit{output labels} \\ \delta_i : Q_i \times Q_i \times (\times_{I \in \mathcal{I}_i} I) \to [0,1] \quad & \textit{transition function} \\ \lambda_i : Q_i \to (\times_{O \in \mathcal{O}_i} O) \quad & \textit{labeling function} \end{aligned}$$

The following additional properties are part of our definition for DBNs.

Property 1. $\cup_{i=1}^{N} \mathcal{O}_i = \mathcal{L}$ *and* $\mathcal{O}_i \cap \mathcal{O}_j = \phi$ *for* $i \neq j$*, meaning that* $\mathcal{O}_1 \ldots \mathcal{O}_N$ *forms a partition of* $\mathcal{L}$*.*

Property 2. *For* $i = 1 \ldots N$*,* δ_i *specifies proper input dependent transition functions. For each* $q \in Q_i$ *and for each input vector* $\mathbf{z}$ *in* $\times_{I \in \mathcal{I}_i} I$

$$\sum_{q' \in Q_i} \delta_i(q, q', \mathbf{z}) = 1. \tag{1}$$

[3]If we let $\mathcal{I}_i = \{I_1, \ldots, I_K\}$ then $\times_{I \in \mathcal{I}_i} I$ is an abbreviation for $I_1 \times \ldots \times I_K$ where we assume any ambiguity regarding the ordering of inputs and outputs is taken care of, for example, by predefining a total ordering on the sets $\Sigma_1 \ldots \Sigma_M$.

This property ensures that transition matrices are well defined. One might think of the input $\mathbf{z}$ as choosing a transition matrix.

Property 3. *Finally, assume w.l.o.g that $\mathcal{I}_i \cap \mathcal{O}_i = \phi$ for each $i = 1 \dots N$, meaning the outputs of a machine are not fed back into itself.*

Note that the topology of the communication graph is implicitly defined by the label structure. By Property 1 above, every output is associated with a unique machine so a mapping $m : \mathcal{L} \to \{P_1, \dots, P_N\}$, associating labels to machines outputting them, can be uniquely determined. Let $P \in \{P_1, \dots, P_N\}$ be a machine with input label set $\mathcal{I} = I_1, \dots, I_K$. Then there is a directed link from machine P to $m(I_k)$ for each $k = 1 \dots K$. Since the direction of causality does not simplify inference calculations[4], we ignore the direction and consider only undirected links in communication graphs.

Observe that the inputs $\mathcal{I}_i$ do not necessary cover $\mathcal{L}$. The labels can be used to encode arbitrary information about the system. In a later example, we show how these labels can be used to compute costs (see Section VI-A).

We will define some basic operations here that will be used later.

Definition 2. *Let $\Pi = \langle \mathcal{L}, P_1, \dots, P_N \rangle$ be a DBN and let $P = \langle Q, \mathcal{I}, \mathcal{O}, \delta, \lambda \rangle$, where $\mathcal{I} \subseteq \mathcal{L}$ and $\mathcal{O} \cap \mathcal{L} = \phi$, satisfy Property 2 and 3 of Definition 1. The **concatenation of a component**, P, onto Π, written ΠP, is defined as*

$$\Pi P = \Pi' = \langle \mathcal{L} \cup \mathcal{O}, P_1, \dots, P_N, P \rangle .$$

It is clear that Π' satisfies the properties of a DBN because 2 and 3 are independently satisfied by all the component machines and since $\mathcal{O} \cap \mathcal{L} = \phi$, the outputs still form a partition of the label space $\mathcal{L} \cup \mathcal{O}$.

Definition 3. *Let $\Pi = \langle \mathcal{L}, P_1, \dots, P_N \rangle$ be a DBN. Define the **removal of a component**, written Π / P_i, where $i \in \{1 \dots N\}$ as*

$$\Pi / P_i = \Pi' = \langle \mathcal{L} \setminus \mathcal{O}_i, P_1, \dots, P_{i-1}, P_{i+1}, \dots, P_N \rangle .$$

Removal of components will cause the DBN to become ill-formed if another component of Π has an input that is an output of the removed component.

Definition 4. *Let $\mathbf{z} \in \times_{\Sigma \in \mathcal{Z}} \Sigma$ be a vector where $\mathcal{Z}$ is a set of alphabets. Define the **vector projection** of $\mathbf{z}$ onto $\mathcal{X} \subseteq \mathcal{Z}$, written $\mathbf{z}|_{\mathcal{X}}$, as just those components of $\mathbf{z}$ that correspond to symbols from the alphabets contained in $\mathcal{X}$.*

It is evident from this definition that $\mathbf{z}|_{\mathcal{X}} \in \times_{\Sigma \in \mathcal{X}} \Sigma$. Also note that this definition has no meaning if $\mathcal{X} \nsubseteq \mathcal{Z}$.

Definition 5. *Let $\mathbf{x} \in \times_{\Sigma \in \mathcal{X}} \Sigma$ and $\mathbf{y} \in \times_{\Sigma \in \mathcal{Y}} \Sigma$ be such that $\mathcal{X}$ and $\mathcal{Y}$ are both sets of alphabets where $\mathcal{X} \cap \mathcal{Y} = \phi$. Define the **vector composition** of $\mathbf{x}$ and $\mathbf{y}$, written $\mathbf{x} \otimes \mathbf{y}$ as a vector having as components the union of the components of $\mathbf{x}$ and $\mathbf{y}$.*

Thus $\mathbf{x} \otimes \mathbf{y} \in \times_{\Sigma \in \mathcal{X} \cup \mathcal{Y}} \Sigma$ and this definition has no meaning if $\mathcal{X} \cap \mathcal{Y} \neq \phi$.

In Definitions 4 and 5, the order of the components in the resulting vectors is not given. We assume that it will be clear from the context what the order should be. One technique for achieving this would be to let the alphabets Σ have a predefined total ordering. Since these vectors never use the same alphabet for two components, the ordering of the alphabets induces a unique ordering on the components of the vectors. Using this ordering uniformly everywhere (including function inputs) ensures consistency as long as the domains are matching.

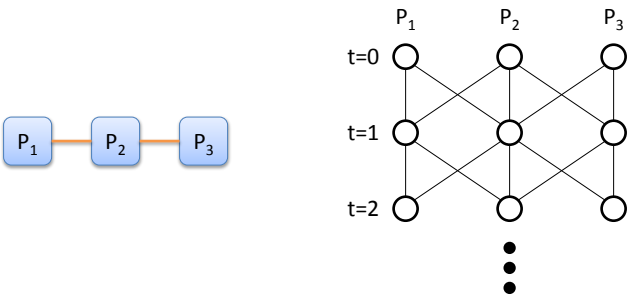
A. Local and Global Views

Figure 2 illustrates the difference between local and global topologies in a DBN. Performing inference on the graph shown in Figure 2(b) can be achieved by using the frontier algorithm which has complexity $\mathbb{O}(TD|Q|^{D+2})$ where T is the number of timesteps, D is the size of the *linear*[5] network and $|Q|$ is the number of states that each node has [6]. Linear complexity in the number of timesteps is good, but it depends upon reasoning over the product state space of $|Q|^D$ states which could be immense depending on the application. Given the configuration of the global topology, there is no obvious way to reduce this by taking the local topology into account. In the frontier algorithm, the local topology merely modifies the exponent but it is never less than $D + 2$ in connected networks.

In this work, we perform the analysis on the graph shown in Figure 2(a). Although each P_i represents a possibly infinite set of behaviors, P_i is represented by a finite symbolic expression. This decomposition should allow us to take greater advantage of the local topology which becomes coupled in the standard decomposition. The remainder of this section will describe how these manipulations are performed.

[4]The fact that A was caused by B does not mean that A carries no information about B.

[5]This formula only describes line networks and will change depending on the local topology. The worst case topology has a complexity of $\mathbb{O}(TD|Q|^{2D})$.



(a) Local topology of a simple DBN.

(b) This is the global topology of the same DBN as in (a).

Fig. 2. (a) shows the local topology. P_1 and P_2 communicate by some shared variables as do P_2 and P_3. However, P_1 communicates only indirectly with P_3 through P_2. (b) is the global topology of the same DBN as in (a). The graph may extend infinitely towards increasing time. Causality goes from top to bottom, but inference can go upstream so the links are treated as undirected. The global view shown in (b) is a detailed view of the behavior of the network depicted in (a). We are interested in using the local decomposition, as in (a), as a basis for computation. In this example, P_1 is conditionally independent of P_3 given P_2.

B. Composition

It is straightforward to define a composition operator that combines two communicating Markov chains. We shall see that the composition still respects Definition 1 so DBNs are closed under composition.

Definition 6. *Let $\Pi = \langle \mathcal{L}, P_1, \ldots, P_N \rangle$ be a DBN and let P_i, P_j be two components of P such that $i \neq j \in \{1 \ldots N\}$. Define the **composition of two DBN components**, P_i and P_j, written $P_i \otimes P_j$ as follows. Let*

$$P_i \otimes P_j = P' = \langle Q', \mathcal{I}', \mathcal{O}', \delta', \lambda' \rangle$$

where

$$\begin{aligned} Q' &= Q_i \times Q_j \\ \mathcal{I}' &= (\mathcal{I}_i \cup \mathcal{I}_j) \setminus \mathcal{O}' \\ \mathcal{O}' &= \mathcal{O}_i \cup \mathcal{O}_j \\ \delta'((q_i, q_j), (q_i', q_j'), \mathbf{z}) &= \begin{array}{l} \delta_i(q_i, q_i', (\mathbf{z} \otimes \lambda_j(q_j))|_{\mathcal{I}_i}) \\ \cdot \delta_j(q_j, q_j', (\mathbf{z} \otimes \lambda_i(q_i))|_{\mathcal{I}_j}) \end{array} \\ \lambda'(q_i, q_j) &= \lambda_i(q_i) \otimes \lambda_j(q_j) \end{aligned}$$

Lemma 1. *Composition by Definition 6 results in a well formed DBN component. (See A for proof.)*

C. Projection

Projection is an operation associated with abstraction and information hiding.

Definition 7. *Let $P_i = \langle Q_i, \mathcal{I}_i, \mathcal{O}_i, \delta_i, \lambda_i \rangle$ and let $\mathcal{X} \subseteq \mathcal{L}$ be such that $\mathcal{X} \cap \mathcal{I}_i = \phi$. Define the **machine projection** of P_i to $\mathcal{L} \setminus \mathcal{X}$, written $\bigoplus_{\mathcal{X}} P_i$ as $P_i' = \langle Q_i, \mathcal{I}_i, \mathcal{O}_i \setminus \mathcal{X}, \delta_i, \lambda_i|_{\mathcal{X}} \rangle$. Here, $\lambda_i|_{\mathcal{X}}$ is the composition of λ_i and the vector projection operator.*

Since only the outputs $\mathcal{O}_i$ and λ_i are affected by this operator, the machine remains well formed. This operation is invalid if $\mathcal{X} \cap \mathcal{I}_i \neq \phi$.

Definition 8. *Let $\Pi = \langle \mathcal{L}, P_1, \ldots, P_N \rangle$ be a DBN and let $\mathcal{X} \subseteq \mathcal{L}$ be such that $\mathcal{X} \cap \mathcal{I}_i = \phi$ for $i = 1 \ldots N$. Define the **network projection** of Π to $\mathcal{L} \setminus \mathcal{X}$, written $\bigoplus_{\mathcal{X}} \Pi$ as applying the machine projection to each component machine, so $\bigoplus_{\mathcal{X}} \Pi = \langle \mathcal{L} \setminus \mathcal{X}, \bigoplus_{\mathcal{X}} P_1, \ldots \bigoplus_{\mathcal{X}} P_N \rangle$ where the individual component $\bigoplus_{\mathcal{X}} P_i$ projections, for $i = 1 \ldots N$ are as described in Definition 7.*

Definition 8 is the extension of Definition 7 to networks. This operation is invalid if $\mathcal{X} \cap \mathcal{I}_i \neq \phi$ for any $i = 1 \ldots N$. None of the inputs to DBN in this definition is exogenous. This means that every input in the DBN specifies an interaction between two of the components. Note that it is possible to eliminate inputs by composition as described in Definition 6. The resulting $\mathcal{I}'$ subtracts the outputs from the union of the inputs meaning that if you compose a machine having an input Σ with another machine having the output Σ, Σ is removed from the set of inputs of the composite.

D. Queries

Queries will be in the form of projections of the DBN to variables of interest. Without this step, it is unlikely that much reduction will be possible. This includes the possibility of defining an observer machine that produces a new output based on quantities of interest. Then the query could be defined as just the outputs of the observer.

E. Algebraic Interpretation

From the previous sections, queries are defined in terms of projections (with possible added observer machines in the network). Consider a query on a set of alphabets $\mathcal{X} \subseteq \mathcal{L}$. From Definition 8, we know that it is not possible to project out certain variables, namely, if an alphabet, $\Sigma \in \mathcal{X}$, is an input to any machine in Π.

The way to solve this problem is to observe from Definition 6 that it is possible to remove inputs by satisfying them via composition with the machines that provide those inputs as outputs. Formally, let Σ be an alphabet of $\mathcal{I}_i$ that is an input of component P_i. By Property 1 of Definition 1, we know that there is a unique machine P_j producing Σ as an output, where by Property 3, $i \neq j$. Removing P_i and P_j from Π and replacing them with the composite $P_i \otimes P_j$, which we may write as $\Pi' = \Pi / P_i / P_j (P_i \otimes P_j)$, results in a well formed DBN that has one less constraint on projecting out Σ. If other components also have Σ as an input, then those components can be subsequently composed, iteratively, starting with $P_i \otimes P_j$, until Σ no longer occurs at all as an input in the DBN. This allows Σ to be eliminated. We have shown the following.

Lemma 2. *Regardless of whether Σ occurs as an input or output, every component having Σ must be composed before Σ can be projected out.*

What emerges is an elimination ordering problem. There are two ways to view this problem, algebraic and graphical. One way to eliminate all of the inputs of a well formed DBN is by composing all of the machines. This way every input will be satisfied. This problem is structured exactly as if solving a constraint program.

Definition 9. *We define the **constraint hypergraph** (or Gaifman graph) of a DBN, $\Pi = \langle \mathcal{L}, P_1, \ldots, P_N \rangle$ as follows. Let $G = (V, E)$ be the hypergraph with nodes $V = \mathcal{L}$ and hyperedges $E = \{\mathcal{I}_i \cup \mathcal{O}_i : i = 1 \ldots N\}$.*

The nodes of this graph consist exactly of the alphabets and there is a hyperedge for every DBN component, linking that component's inputs and outputs. The graph structures the possible orderings of compositions and projections. As indicated by Lemma 2, an alphabet Σ can only be eliminated if every component having Σ as an input or output has been composed first. This is equivalent to saying that the nodes of the constraint graph can be eliminated only after all hyperedges linked to that node have been composed. The constraint graph is determined by the DBN and can be seen as varying dynamically as the DBN is manipulated by composition and projection operations.

An important question is, for a given query, what is the optimal way to order the composition and projection operations? A fact to keep in mind is that composition tends to increase the complexity of the model (we take the Cartesian product of state spaces) and projection tends to reduce the complexity (we remove information from the model). Intuitively, we would like to perform as many projections as possible, as early as possible in the sequence. However, due to the condition of Lemma 2, certain compositions must precede those projections. So assuming that we wish to perform the elimination optimally, the projection ordering determines when the compositions are needed. Finding an optimal projection ordering is a difficult problem in general, but for certain classes of graphs, such as chordal graphs or trees, it can be done in linear time (this is reviewed in [2]).

F. Reduction

There is one last operation that is used, typically in conjunction with the projection operator, which is reduction. In the case of state machines, an efficient, $\mathbb{O}(nlogn)$, algorithm was discovered by Hopcroft [7]. It turns out that generalizing this same algorithm to Markov chains is possible [8]. It is also possible to implement this algorithm using only simple data structures and algorithms [9]. See Figure 3 for an example.

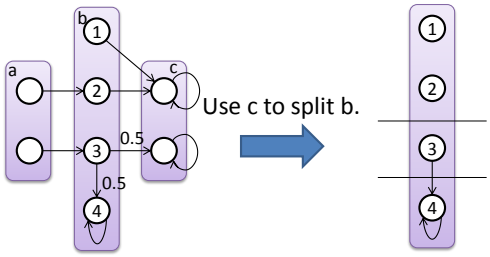


Fig. 3. Illustrating a key step in the working of the lumping algorithm. The algorithm begins with the coursest possible partitioning of the states and incrementally refines by splitting. In this example, states 1,2 lead to partition c with probability 1, state 3 leads to partition c with probability 0.5 and state 4 leads to partition c with probability 0. This means that partition b is refined into the partitions $\{1, 2\}$, $\{3\}$ and $\{4\}$. Essentially this splitting of an existing partition based on the probability of reaching another partition continues until convergence. The log complexity comes from the fact that whenever you refine a partition, one of the new resulting partitions is redundant for splitting purposes.

In this work, we generalize this algorithm further to operate on DBN components. The tricky part of this is that DBN components do not have closed form transition functions, but are actually functions of their input variables $\mathcal{I}$. In a normal Markov chain, $\delta(q, q')$ is given by a fixed probability. However, in a DBN component, $\delta(q, q', \mathbf{z})$ is parameterized. The lumping

algorithm, which works by partition refinement requires that we can sort the transition probabilities. In this case, the transition probability is a symbolic expression. As long as there is a canonical form for this symbolic expression, then it can be *lexically* compared to other symbolic expressions. A canonical, unique form is needed because otherwise, there could be two symbolic expressions describing the same function that are lexically unequal. This canonical form for transition functions is described in Section III-C.

III. Multiple Decision Diagram (MDD)

We use MDDs (see [10]) to symbolically encode the transition functions in this problem. MDDs are very similar to the well known binary decision diagrams (BDDs) with operations described by Bryant [11]. The only difference is that rather than two possible choices at each node, there are many possible choices and there are also many possible terminals. Using MDDs instead of BDDs allows us to bypass a binary encoding step for the variables. The operations are nearly identical to operations on BDDs. We assume that there is a given variable ordering and do not concern ourselves with dynamic variable reordering although this is something that could be considered in future work.

A. Operations

We implement the following operations on MDDs. In this context, it is sufficient to treat the MDDs as mappings, $m : \mathcal{L} \to T$, where T is some arbitrary codomain. Note that this same descriptor applies even in cases where the actual domain of the MDD is a subset of $\mathcal{L}$ as the extraneous variables can be ignored.

- apply This takes as parameters two MDDs, $m1 : \mathcal{L} \to T_1$ and $m2 : \mathcal{L} \to T_2$ and a mapping from the respective MDD terminal types to an arbitrary output type, $f : T_1 \times T_2 \to T_3$ where T_1, T_2 and T_3 are not necessarily different. The operation effectively returns an MDD describing $f(m1(\mathbf{z}), m2(\mathbf{z}))$.
- sum This takes as parameters an MDD $m : \mathcal{L} \to T$, an alphabet $\Sigma \in \mathcal{L}$ in the domain of the MDD and the definition of a summation operator $s : T \times T \to T$. If we let $\Sigma = \{\sigma_1, \ldots, \sigma_K\}$, then this operator returns $s(\ldots s(s(m|_{\sigma_1}, m|_{\sigma_2}), m|_{\sigma_3}), \ldots, m|_{\sigma_K})$.
- restrict This operation takes as parameters an MDD and a particular assignment to one of the variables of that MDD. It returns an MDD that is equal to the input MDD restricted to that variable assignment.
- remap This operation takes as parameters an MDD, $m : \mathcal{L} \to T$ and a mapping $f : T \to T'$. It returns the composition $f \circ m$.

These operations all run in time proportional to the size of the input MDDs.

B. Implementation of Natural Joins and Aggregations

One feature of MDDs is that it is trivial to implement *weighted natural joins* using them. In fact, using the terminal nodes as weights and invoking *apply* on two MDDs using the appropriate semiring multiplication operator exactly produces the natural join. As this implementation of MDDs also includes the *summation* over a variable, using an appropriate semiring sum operator means that a full summary propagation inference engine can be constructed from MDDs alone. This means that many problems such as Boolean satisfiability, inference on Bayesian networks and optimization using $\max +$ algebra can use MDDs as a solver [2].

C. Encoding of Transition Functions

Every state of a DBN component is associated with a transition vector which assigns probabilities to next states. The transition vector is not a vector of probability values, but rather a function mapping from outputs of neighboring machines to probability vectors. The MDD encodes a decision tree based on the discrete values of the labels of the inputs. Since like BDDs, MDDs are canonical, this is sufficient for providing a canonical representation for the purposes of comparing two symbolic transition probabilities. To order them, we flatten the MDD by depth first recursion and compare the resulting strings (this is done incrementally so that the full strings need not be generated if inequality is detected early in the recursion).

D. Implementation of Composition

Composition of DBN components, as described in Definition 6, requires the multiplication of transition functions. This is achieved by two MDD operations, the first being *restrict*, which is used to feed the output of one MDD to another. There is a different transition function encoded for every starting state. In the product state (q_i, q_j), the outputs $\lambda(q_i)$ and $\lambda(q_j)$ are fixed, so they can be made constants in the resulting transition function (if they occur as inputs) and eliminated using *restrict*. *Apply* is then used with something resembling an outer product of transition vectors as the mapping parameter.

Since projection only applies when the alphabets do not occur as inputs, it does not require an MDD operation.

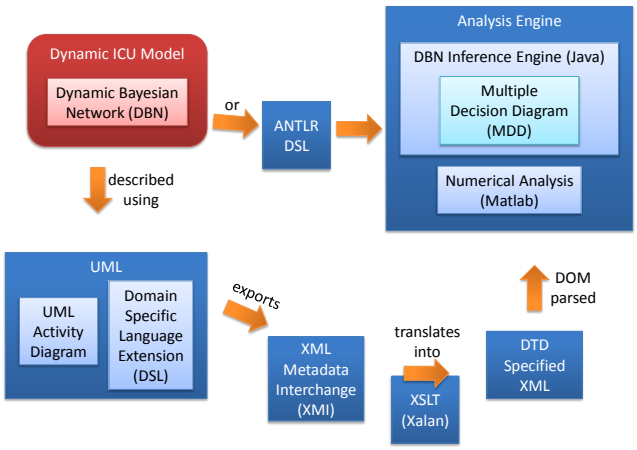


Fig. 4. The above figure shows our toolchain. The DBN is specified using either extended UML activity diagrams or using a DSL implemented with ANTLR. We use the export function of Papyrus to produce XMI which is then translated using XSLT to another format for our tool.

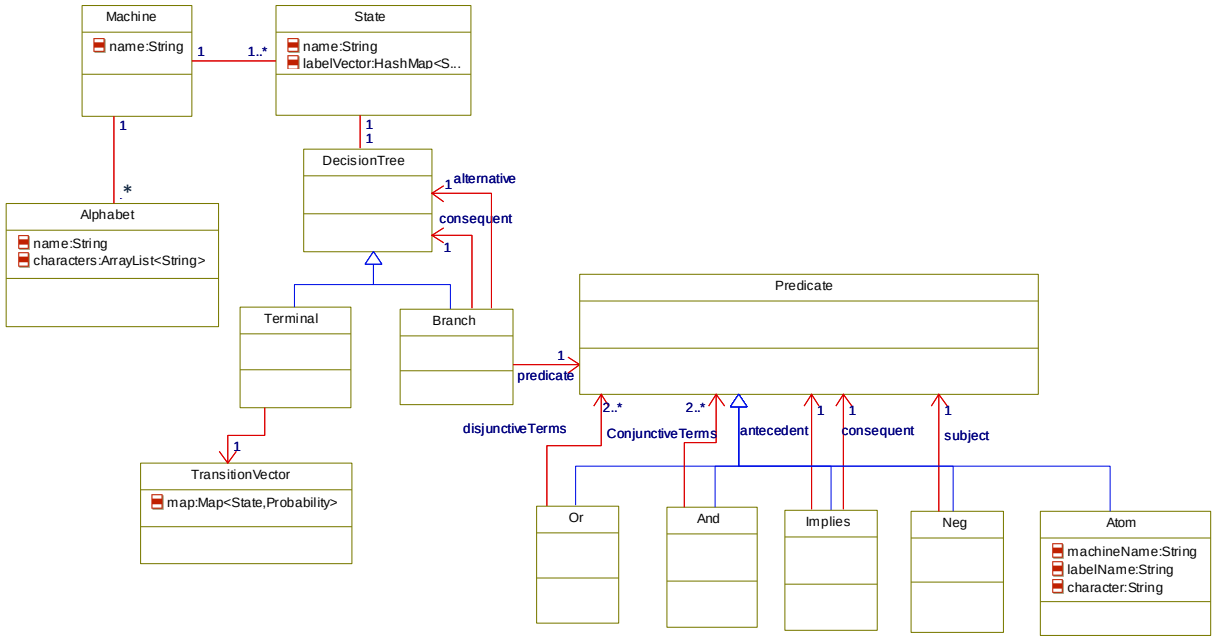


Fig. 5. The class diagram for the DBN components as they are entered by the user. Observe the recursive links in `DecisionTree` and `Predicate`. These are both implementations of the *composite* pattern used to represent trees. `DecisionTree` captures arbitrarily nested `if then else` constructs. Likewise, `Predicate` captures arbitrarily nested logical predicates consisting of expressions connected by logical operators such as `AND`, `OR` or `NOT`.

IV. Tool Implementation

The tool has been implemented in Java on top of some other technologies as shown in Figure 4. Linking to UML provides us with a rich graphical framework of modeling primitives and we eventually forsee linking to other languages that can link to UML such as Modelica. Using XSLT as a parser provides a simple mechanism for linking to multiple UML tools with slightly different XMI dialects (we have implemented translations from both Papyrus and UML2 Tool). We use Xalan as the XSLT transformation language which allows Java based extensions. The intermediate XML format is used to provide an independent storage format for our models that can easily be used by other tools. We also have a way to directly express the entire model using a domain specific language described in Section IV-A.

A. Specification Language

Figure 5 shows the structure, as presented to the user, of the multiple `Machine` instances comprising the components of a DBN. λ, the output labeling function, is captured by a `HashMap` that contains entries mapping the `String` names of each alphabet $\Sigma \in \mathcal{O}$ to the approriate `String` names of each symbol in Σ.

The transition functions δ are modeled using `if then else` constructs with predicates on the inputs $\mathcal{I}$ of the DBN components. The terminals of these expressions are the transition vectors. This mechanism provides a relatively compact,

```
Machine: p1
  State: using  labels: using=1
    if      commons::level=0 p[resting]=1.0, p[using]=0.
    else if commons::level=1 p[resting]=0.8, p[using]=0.2
    else if commons::level=2 p[resting]=0.6, p[using]=0.4
    else if commons::level=3 p[resting]=0.4, p[using]=0.6
    else if commons::level=4 p[resting]=0.2, p[using]=0.8
    else    // commons::level=5
      p[resting]=0,   p[using]=1
  State: resting  labels: using=0
    p[resting]=0.5, p[using]=0.5
```

Fig. 6. This is a a very simple two state example of DBN component specification written in our DSL just to show the syntax. The two states are `using` and `resting`. The `if then else` construct is used to describe $\delta(\texttt{using}, q', \mathbf{z})$ The predicate conditions refer to the outputs of another machine called `commons`. The terminals of these decision trees are transition vectors. Consider this in relation to Figure 5.

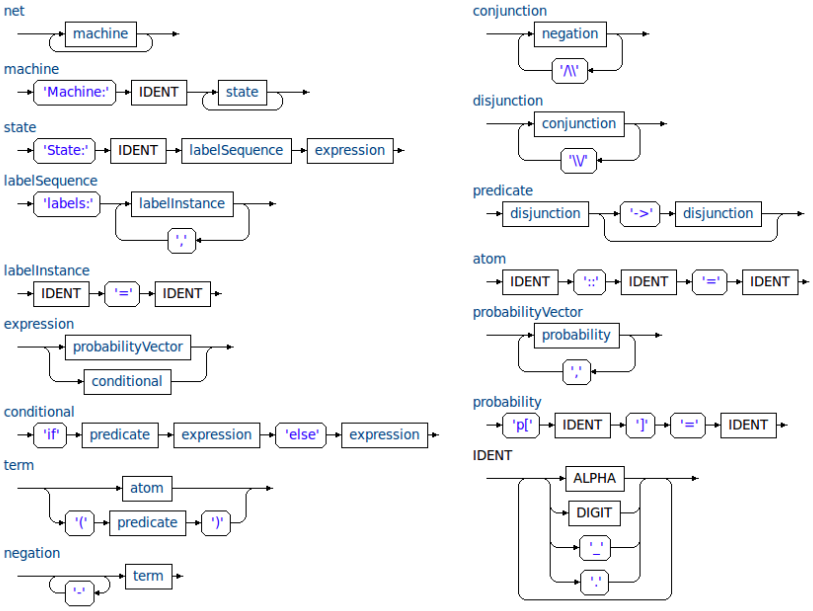


Fig. 7. This figure shows the specification of the ANTLR grammar used to describe DBNs. The toplevel grammar production is a `net` which consists of a number of machines. You will note that $\mathcal{L}$ and the various $\mathcal{O}$ and $\mathcal{I}$ values are not explicitly defined anywhere in this grammar, but can be inferred by the parser. This grammar does not have the dangling else problem because every `if` must have a corresponding `else` so extra delimiters are unnecessary.

relatively easy to use means to describe arbitrary mappings $\delta(q, q', \mathbf{z})$. The input q is determined by the particular state, then a transition vector is used to cover all the possible q' next states. The discrete label space $\mathcal{I}$ is broken up using logical predicates whose atoms are label assignments. See Figure 6 for an example.

One thing to point out here is the naming scheme. Due to Property 1, every alphabet $\Sigma \in \mathcal{L}$ is initially output by a unique DBN component P. In our implementation, every such initial component is given a name by the user so that it can be referred to in other places in the specification. This name is used as a prefix for the names of each Σ generated by P. We use the scope resolution operator (`::`) to separate the prefix from the actual name. This helps the user with awareness of where the symbols are coming from and helps with naming for things like symmetrical instances.

The code shown in Figure 6 is parsed by ANTLR [12], a lexer parser generator for Java, having the specification shown in Figure 7. There are a few rules not shown having to do with comments and whitespace.

1) Code Generation: In the system we were analyzing (see Section VI-A), this particular grammar was found to be too restrictive for practical use and we had to use code generation to produce some of the machine specifications. In particular, the DBN component responsible for randomly distributing arrivals to unoccupied beds had extremely large decision trees. The reason for this is the following. Every configuration of empty vs occupied beds (of which there are 2^N where N is the size of the ICU), requires a different transition vector. This makes the resulting MDD representing the transition function very large. We are considering what might make sense in the next iteration of this tool as higher level constructs for creating these specifications.

B. Implementation Style

The MDDs, though written in Java are implemented using a completely functional coding style by liberal use of `final` modifiers on members and enforcement through interfaces. The key bottlenecks in our code, computing symbolic transition matrices, and computing compositions of DBN components[6] have been parallelized by exploiting this fact. We are free to perform whatever manipulations we wish in parallel on the MDDs because they are decoupled by their immutability.

C. Monte Carlo

The tool also contains a very simple implementation of Monte Carlo simulation. This was done to verify the results of the symbolic analysis. The Monte Carlo simulation uses a similar query framework. It is possible to define a query as a new DBN observer component to be concatenated with Π and project onto just the output variables of the observer. If we were to try to collect statistics over the entire label space, we would run into a curse of dimensionality. So the query framework still turns out to be quite useful. We do not, however, do any local topology based analysis using Monte Carlo, which could conceivably be useful even in the simulation context.

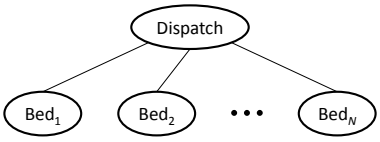
V. Related Work

This work is somewhat similar to the work in the probabilistic model checker Prism [13] and also the tool Mobius [14]. What makes our tool quite distinct from these two is its utilization of *projection* as an integral technique in its solver.

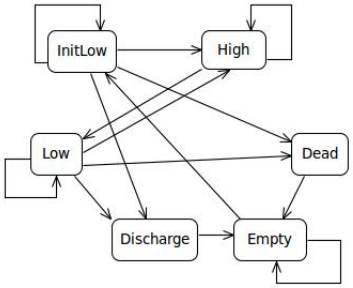
VI. Results

A. Intensive Care Unit (ICU) Modeling

We model the ICU as the simple system depicted in Figure 8(a).



(a) This is the local topology of a DBN modeling the ICU.



(b) This figure shows the patient progression model as entered into Papyrus.

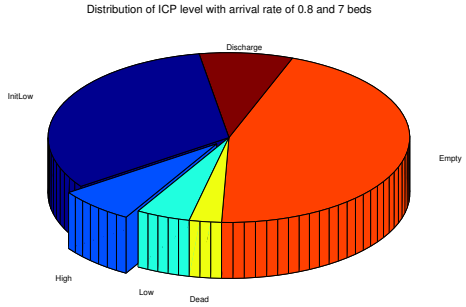
Fig. 8. As shown in (a), each $\texttt{Bed}_i$ represents a bed of the ICU. Each $\texttt{Bed}_i$ captures the patient progression model and uses an additional empty state to indicate an unoccupied bed. The `Dispatch` component encompasses the arrival of patients and placing them in an unoccupied bed or blocking if no bed is available. As shown in (b), a detailed specification starting from the `conditional` of the grammar is contained within each node. The patients enter in an initially low intracranial pressure (ICP) state, and potentially reach a high ICP state. They may return to a high ICP state. Every patient eventually leaves the ICU by either death or discharge.

We have a very simple patient progression model of severe traumatic brain injury (STBI) patients shown in Papyrus in Figure 8(b). The problem of modeling ICUs has been studied previously by simulation methods [15]. The analysis performed here could also be performed using Monte Carlo simulation, however, it would then be necessary to incorporate an additional error term in the calculations. The emphasis in this study was on developing compositional methods of analysis.

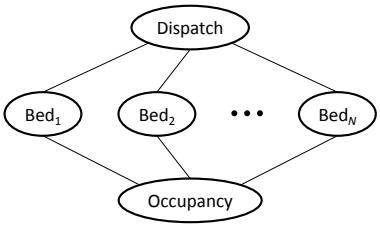
Based on documents from the UMMC Shock Trauma center [16], we were led to understand that management of intracranial pressure (ICP) is one of the most important aspects of treating STBI patients. The model presented focuses only on this variable although there are certainly other criteria that may influence a patient progression. Associated with each of the states is a different cost reflecting the cost of different clinical treatments depending on the ICP level.

We use our model to measure the overflow probability, the occupancy and the expected cost.

The strategy for querying overflow probability is to observe the `Dispatch` component. We compose the $\texttt{Bed}_i$ components with `Dispatch` one at a time and project out any variables having to do with $\texttt{Bed}_i$. What we are left with is a single bit we are observing, which represents arrivals of "things". Since we know the arrival rate, we can calculate the overflow probability by comparing the observed probability of arrival to the actual arrival rate.



(a) The stationary distribution of states for one bed in the ICU.

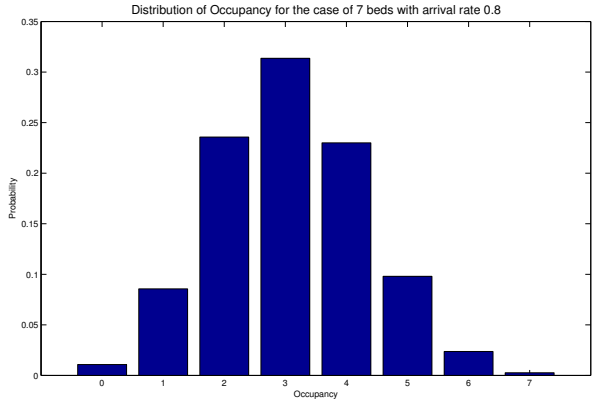


(b) The local structure of DBN.

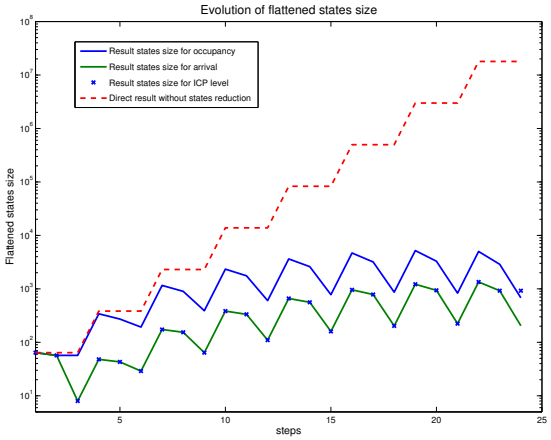
Fig. 9. (a) Shows the stationary distribution of states for one bed in the ICU. Due to symmetry, the distribution of states for the other beds is identical and the cost can be computed based on what it costs to be in each state. (b) Shows the local structure of the DBN used to query occupancy. An observer is added to measure occupancy. We can tell by the structure of this graph that the observer will add complexity to the system.

To query the expected cost, we compose the $\texttt{Bed}_i$ components with `Dispatch` one at a time and project out any variables having to do with $\texttt{Bed}_i$ until the last $\texttt{Bed}_N$ component. At this point, we sum everything out except for the costs. By symmetry, multiplying this by N gives the overall ICU cost. Figure 9(a) shows the distribution of ICP state which is a proxy for the cost.

Since occupancy has an effect on cost per patient in the form of supplementary nurse costs [17], it is interesting from a cost modeling perspective. To query the occupancy, we must create an observer machine, as shown in Figure 9(b). Figure 10(a) shows the resulting occupancy. The occupancy will be given by first taking the product of `Occupancy` and `Dispatch` then incrementally taking the product with each $\texttt{Bed}_i$ and projecting out any variables that are not occupancy.



(a) The stationary occupancy distribution



(b) State size over time.

Fig. 10. (a) Showns the stationary occupancy distribution as observed by the occupancy observer shown in Figure 9(b). (b) Shows the number of states as a function of the number of steps in the inference. The dashed red curve shows how many states there would be if we were taking a raw Cartesian product at each step. The other curves show the number of states for different queries. The sawtooth pattern seen is a result of the project compose pattern. Projection is followed by reduction which reduces the number of states and composition causes the number of states to increase. The small dip seen after each composition is caused by removing unreachable states.

Figure 10(b) shows the state reduction achieved in these queries. As anticipated, the occupancy query is slightly more complex than the others. The blocking probability query and the cost query are identical except in the last step because the queries are the same except that the blocking probability query projects out the last patient's cost information while the cost query does not.

VII. Discussion

The primary goal of this study was to investigate complexity reduction using local tree structures of systems. Ferrara [5] shows that it is possible to encode exponential space Turing machines using bounded communicating automata over a treewidth bounded communication topology. This means that communicating automata, even restricted to bounded treewidth communication topologies, are powerful enough to encode arbitrary EXPSPACE computations. However, this is similar to the situation with NP-complete problems where in general, certain types of computations can be used to encode general NP-complete problems. The difference is that in the case of NP-complete, treewidth alone is sufficient for complexity reduction.

[6]Lumping of the Markov chain fragments runs relatively fast compared to these steps.

Ferrara's proof shows that treewidth alone is not sufficient for complexity reduction in the DBN case, but in the systems analyzed, significant reductions in complexity were achieved.

A detailed look at the systems studied reveals a certain symmetrical structure that helped a great deal in reducing the states. The patient beds are all essentially the same DBN component, replicated with different names. When the identity specific labels are projected out, it becomes possible for the lumping algorithm to detect the symmetry and reduce. The specific symmetry found is that of order invariance. The state needed to represent N automata with Q states each is simply a count of the number of automata in each state. Counting the number of possible configurations is equivalent to counting the number of ways N elements can be interleaved in a sequence with $Q-1$ elements (only $Q-1$ separators are needed). This is known to be $\binom{N+Q-1}{N}$ which is significantly less than Q^N. While in retrospect, we can know this a priori and next time design a tool to exploit this symmetry immediately, it is an interesting result that symbolic Markov chain lumping alone was able to detect and reduce such symmetries.

A critical question is whether the low treewidth helped at all in this analysis. The answer is undoubtably yes because without using the incremental approach of summary propagation, we might have needed the entire state space to be constructed prior to passing it to the Markov chain lumping algorithm. This would have been infeasible because of the size of these product state spaces. What projection-composition does in this case is that it feeds the lumping algorithm only smaller, feasibly sized pieces of the problem to work on.

The ultimate computational obstacle in this exercise was not the number of states, but the rapidly expanding MDDs needed to fully describe the rather complex decision trees. This will be a good topic for future research.

VIII. Limitations and Future Work

As preliminary work, there are a number of questions that we would like to investigate in the future.

A. High Level Language

As described in Section IV-A1, the language presented is too low level for effectively describing certain complex transition functions. While we are currently using Java to perform code generation for these special cases, it should be possible to extract some key operations out from the Java code to extend the language so that it becomes higher level. One possibility is to start with a high level language like Java, place some restrictions on it and some extensions for probabilistic behavior, then perform translation to a lower level representation.

B. Symmetry Reductions

If symmetry is such a strong contributor to state reduction, perhaps it will be symmetry that is required for efficient analysis in large systems with local topologies. We will be working on this question in the future.

IX. Conclusions

We have created a tool that analyzes DBNs based on local topology. This question has significance to the systems engineering community because it addresses the question of scalable compositional reasoning and deepening our understanding of block diagrams. We achieve a local topology based analysis by using a symbolic representation of the transition functions. The projection composition framework is a generalization of the sum-product algorithm, which is typically used on commutative semirings. Since sums in the sum-product algorithm are only used to sum out a variable, slightly less powerful objects than semirings are needed to run the algorithm. We use projection here as a unary operator that serves as an analogy for summation. Composition is a very natural analogy for multiplication.

The interesting question is whether using the local topology helps in reducing the complexity of analysis. In the examples that we have analyzed, this in indeed the case, even though the main reduction is coming from symmetry. This shows that state minimization is a powerful technique that includes symmetry reduction and we have also shown that it can be made to run incrementally, following the local topology, which makes it possible to analyze state spaces that we believe would be intractable to represent.

The main reductions from symmetry, if applied upfront, could be more effective than this approach and it is something we intend to explore further.

Appendix

Property 1 still holds because P' replaces P_i and P_j merging $\mathcal{O}_i$ and $\mathcal{O}_j$. This means that the union is still $\mathcal{L}$ and intersections are still disjoint.

To see that Property 2 holds for P', it suffices that show that for every starting state (q_i, q_j) and every input $\mathbf{z} \in \mathcal{I}'$, summing δ' over the possible next states (q'_i, q'_j) results in 1.

$$\forall (q_i, q_j) \forall \mathbf{z} \sum_{(q'_i, q'_j)} \delta'((q_i, q_j), (q'_i, q'_j), \mathbf{z}) = 1 \tag{2}$$

For each starting state (q_i, q_j), $\lambda_j(q_j)$ and $\lambda_i(q_i)$ are constant implying that for a given $\mathbf{z}$, $\mathbf{z}_i \equiv \mathbf{z} \otimes \lambda_j(q_j)|_{I_i}$ and $\mathbf{z}_j \equiv \mathbf{z} \otimes \lambda_i(q_i)|_{I_j}$ are both constant. It suffices to show that for a given constant $\mathbf{z}_i$ and $\mathbf{z}_j$ that

$$\sum_{(q_i', q_j')} \delta_i(q_i, q_i', \mathbf{z}_i) \cdot \delta_j(q_j, q_j', \mathbf{z}_j) = 1. \tag{3}$$

Since q_i' and q_j' do not occur as a tuple in 3, we can split the sum and collect like terms resulting in the equivalent expression

$$\sum_{q_i'} \delta_i(q_i, q_i', \mathbf{z}_i) \cdot \sum_{q_j'} \delta_j(q_j, q_j', \mathbf{z}_j) = 1. \tag{4}$$

We know from (1) that both of the above factors equals 1 so the result holds.

That Property 3 holds is trivial based on the definition of $\mathcal{I}'$ as explicitly excluding alphabets from $\mathcal{O}'$.

ACKNOWLEDGMENT

The authors would like to thank Jiban Khuntia, Adam Montjoy, Jeffrey Herrmann, Guodong (Gordon) Gao and Ritu Agarwal for their assistance in creating the ICU model.

Research supported in part by the NSF under grant CNS-1035655, by the NIST under contract 70NANB11H148 and by a grant from the Lockheed Martin Corporation.

REFERENCES

[1] M. Fowler, *Refactoring: improving the design of existing code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.
[2] S. Yang and J. Baras, "Factor join trees for systems exploration," in *Proceedings 23rd International Conference on Software & Systems Engineering and their Applications*, Paris, France, 2011.
[3] S. Arnborg and A. Proskurowski, "Linear time algorithms for NP-hard problems restricted to partial k-trees," *Discrete Applied Mathematics*, vol. 23, no. 1, pp. 11–24, 1989.
[4] K. Murphy and Y. Weiss, "The Factored Frontier Algorithm for Approximate Inference in DBNs," in *Proceedings of the Seventeenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*. San Francisco, CA: Morgan Kaufmann, 2001, pp. 378–385.
[5] A. Ferrara, G. Pan, and M. Vardi, "Treewidth in verification: Local vs. global," in *Logic for Programming, Artificial Intelligence, and Reasoning*. Springer, 2005, pp. 489–503.
[6] K. Murphy, "Dynamic bayesian networks: Representation, inference and learning," Ph.D. dissertation, University of California, Berkeley, 2002.
[7] J. E. Hopcroft, "An n log n algorithm for minimizing states in a finite automaton," Stanford, CA, USA, Tech. Rep., 1971.
[8] S. Derisavi, H. Hermanns, and W. Sanders, "Optimal state-space lumping in Markov chains," *Information Processing Letters*, vol. 87, no. 6, pp. 309–315, 2003.
[9] A. Valmari and G. Franceschinis, "Simple O (m log n) time Markov chain lumping," *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 38–52, 2010.
[10] D. Miller and R. Drechsler, "Implementing a multiple-valued decision diagram package," in *Proceedings 28th IEEE International Symposium on Multiple-Valued Logic*. IEEE, 1998, pp. 52–57.
[11] R. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys (CSUR)*, vol. 24, no. 3, pp. 293–318, 1992.
[12] T. Parr, *The definitive ANTLR reference: Building domain-specific languages*. Pragmatic Bookshelf, 2007.
[13] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic symbolic model checker," *Computer Performance Evaluation: Modelling Techniques and Tools*, pp. 113–140, 2002.
[14] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. Doyle, W. Sanders, and P. Webster, "The mobius modeling tool," in *Proceedings 9th International Workshop on Petri Nets and Performance Models*. IEEE, 2001, pp. 241–250.
[15] A. Seila and S. Brailsford, "Opportunities and challenges in health care simulation," *Advancing the Frontiers of Simulation*, pp. 195–229, 2009.
[16] "ICP Management Protocol," 2011, private communication by S. Yang.
[17] J. Griffiths, N. Price-Lloyd, M. Smithies, and J. Williams, "Modelling the requirement for supplementary nurses in an intensive care unit," *Journal of the Operational Research Society*, vol. 56, no. 2, pp. 126–133, 2005.