Symbolic Computation and Representation of Deadlock Avoidance Policies for Complex Resource Allocation Systems with Application to Multithreaded Software

Zhennan Fei, Knut Åkesson and Spyros Reveliotis

Abstract— **In our recent work, we proposed a series of binary decision diagram (BDD-) based approaches for developing the maximally permissive deadlock avoidance policy (DAP) for a class of complex resource allocation systems (RAS). In this paper, (i) we extend these approaches by introducing a procedure that generates a set of comprehensible "guard" predicates to represent the resulting DAP, and (ii) we customize them to the problem of deadlock avoidance in shared-memory multithreaded software, that has been previously addressed by the Gadara project. In the context of this last application, the generated guards can be instrumented directly into the source code of the underlying software threads, providing, thus, a very efficient and natural representation of the target policy. At the same time, by integrating the representational and computational strengths of symbolic computation, the presented approach can support the computation of the maximally permissive DAP for RAS corresponding to problem instances of even larger scale and complexity than those addressed in the current literature.**

I. Introduction

In the Discrete Event Systems (DES) literature, deadlock avoidance for sequential, complex resource allocation systems (RAS) is a well-established problem that arises from many contemporary technological applications [1], [2]. In its basic positioning, this problem concerns the coordinated allocation of the system resources to concurrently executing processes so that every process can eventually proceed to its completion. In particular, by utilizing the information about the current allocation of the system resources and the available knowledge about the structure of the executing processes, the applied control policy avoids the visitation of RAS states from which deadlock is inevitable.

The work presented in this paper is an extension and an application of the symbolic framework that has recently been proposed in [3], [4] for the effective and computationally efficient development of the maximally permissive deadlock avoidance policy (DAP) for various RAS classes. More specifically, by modeling any given RAS instance in the modeling framework of the *extended finite automata* (EFA), the approaches of [3], [4] employ several *binary decision diagram* [5] (BDD) based algorithms for symbolically computing the target DAP. Besides the employment of symbolic computation, additional efficiencies for the aforementioned algorithms are obtained from the fact that they avoid the complete exploration of the underlying RAS state-space. This capability is established upon the crucial fact that, in the considered RAS dynamics, unsafety is defined by inevitable or uncontrollable adsorption into the system deadlocks. Therefore, the target unsafe states can be retrieved by a computation that starts from the RAS deadlocks and "backtraces" the RAS state-space until it hits the boundary between the safe and unsafe subspaces. The unsafe states that are directly accessible from the safe subspace are known as the "boundary" unsafe states of the state-space. Furthermore, the entire set of boundary unsafe states can be effectively represented by its minimal elements since the notion of unsafety presents a monotonicity property that endows this set with properties similar to those of a right-closed set [6].

The availability of the set of minimal boundary unsafe states that is computed from [3], [4] enables an expedient one-step-lookahead scheme preventing the RAS from reaching outside its safe region. In particular, any tentative transitions taking the underlying RAS to a state that dominates, component-wise, some minimal boundary unsafe state will be disabled by the target DAP. The BDD-based symbolic representation of the set of minimal (boundary) unsafe states offers a compact and operationally efficient way to deploy the target DAP. However, such a symbolic representation of the forbidden states is of very limited comprehensibility. Furthermore, the resultant control policy is of very centralized nature, a feature that can be deemed as limiting/undesirable in the context of certain applications.[1] In this work, we seek to address these concerns by expressing the target DAP as a set of comprehensible logic formulas that will further "guard" the underlying resource allocation function.

More specifically, inspired by the work in [7], this paper extends the aforementioned BDD-based approaches by introducing a procedure that generates a set of comprehensible "guard" predicates to represent the resulting DAP. In particular, by attaching these predicates to the original model, we guard against transitions to RAS states that dominate elements of the set of minimal boundary unsafe states. Furthermore, we customize the developed approaches to the problem of deadlock avoidance in shared-memory multithreaded software, which has been previously addressed by the Gadara project [8], [9], [10]. From a more executional standpoint, this customization evolves in the following three stages:

- We re-cast the Gadara nets that model the primitive lock acquisition and release operations of multithreaded programs into equivalent EFA models (Section II).

Z. Fei and K. Åkesson are with the Automation Research Group, Department of Signals and Systems, Chalmers University of Technology, SE-412 96, Gothenburg, {zhennan, knut}@chalmers.se.

S. Reveliotis is with the School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, GA-30332, United States, spyros@isye.gatech.edu.

[1] including the particular application considered in this paper

- We demonstrate how the symbolic algorithms presented in [3] can be adapted and used to compute the set of minimal boundary unsafe states from the EFA model (Section III).
- We present the aforementioned BDD-based guard generation procedure for computing the necessary guard predicates for the RAS-modeling EFA from the BDD representing the set of minimal boundary unsafe states, and we further streamline this procedure by taking advantage of certain RAS properties that are implied by the considered application (Section IV).

The generated guards can be instrumented into the source code of the underlying software threads, providing, thus, a very efficient and natural representation of the target policy. At the same time, by effectively integrating the representational and computational strengths of symbolic computation, the new approach outlined above can support the computation of the maximally permissive DAP for RAS corresponding to problem instances of even larger scale and complexity than those addressed in the current literature.

II. Modeling Gadara Nets as EFA

In this section we use a simple multithreaded program in order to demonstrate how the Gadara net modeling the lock acquisition and release operations in a given program can be re-cast into an equivalent EFA model. We start by reviewing the concept of the Gadara net itself, assuming, however, that the reader is already familiar with the basic Petri net (PN) modeling framework, its major concepts, and the related terminology. An introduction to the PN modeling framework and the relevant theory can be found in [11].

A. Gadara Nets

A Gadara net [9], $\mathcal{N}_G$, is a special class of Petri nets that is used to systematically model the dynamics of the lock acquisition and release operations that take place in multithreaded programs. From a structural standpoint, these nets can be perceived as a set of strongly connected state machines, modeling the critical regions of the various program threads, which interact through a set of common places that model the allocation status of the various program locks.

The state set P_i of each state machine $\mathcal{N}_i$ is partitioned to a singleton $\{p_{0i}\}$ and the set $P_{S_i} \equiv P_i \setminus \{p_{0i}\}$. Place p_{0i} is known as the *idle place* of the subnet $\mathcal{N}_i$, and it models the environment of the corresponding critical region; i.e., tokens in place p_{0i} model thread instances waiting to enter this critical region, or tokens exiting the region. On the other hand, places in P_{S_i} are characterized as the set of *operation places* for the subnet $\mathcal{N}_i$, and they model the various process stages that take place in the critical region. Furthermore, the connectivity of the subnet of $\mathcal{N}_i$ that is obtained from the removal of the idle place p_{0i} and its incident arcs, essentially expresses the sequential logic that defines the evolution of a thread instance while in the corresponding critical region. In a similar spirit, the state machine structure of $\mathcal{N}_i$ expresses the atomic nature of any thread instance executing in a critical region. In the sequel, following standard RAS terminology, we shall also refer to the Gadara subnets $\mathcal{N}_i$ as the model *"process"* subnets.

On the other hand, the set of the interconnecting places of the process subnets that model the lock allocation, will be denoted by P_R, and its elements will be referred to as the *"resource"* places of the Gadara net. From a more theoretical standpoint, the connectivity of a resource place $r \in P_R$ to the net transitions establishes a place invariant for the overall Gadara net that expresses the mutually exclusive and reusable allocation of the corresponding lock. Furthermore, the Gadara model assumes that the lock allocation function does (should) not interfere with the branching logic of the executing processes, and therefore, it forbids any arcs leading from a resource place to a transition that is not the unique option for its input operation place in the corresponding process subnet. In fact, it is also requested that such a decoupling of the control function from the basic sequential logic that drives the process execution within a critical region should also be observed by the sought DAP. From a more technical standpoint, this last requirement is enforced by treating transitions that model branching decisions for the underlying critical region as *uncontrollable* by the sought policy.

Based on the above description of the semantics of the Gadara net, the initial marking $M(p)$ for an operation place p of any subnet $\mathcal{N}_i$ is naturally set equal to zero. On the other hand, the initial marking $M(p_{0i})$ for the idle place of any subnet $\mathcal{N}_i$ is typically set to some sufficiently large positive (integer) value so that this value does not restrict artificially the concurrency in the net dynamics that is naturally enabled by the (uncontrolled) lock allocation function. Finally, in the basic Gadara model, locks correspond to mutexes, and they are distinct entities; therefore, the initial marking $M(r)$ of each resource place r is set equal to one.

An example Gadara net $\mathcal{N}_G$ is depicted in Fig. 1. This Gadara net consists of two process subnets, $\mathcal{N}_1$ and $\mathcal{N}_2$. The set of the operation places and the relevant sequential logic for each process subnet are defined by the corresponding paths depicted in black in the figure. On the other hand, the corresponding idle places and their connection to the rest of the process subnet are depicted in purple. The three *resource places* of the considered net and their connectivity to the rest of the net are respectively shown in blue, red, and green. The reader should notice that the branching transitions t_{13} and t_{16}, shown as the hollow bars in Fig. 1, are modeled as the uncontrollable transitions and their only input place is the operation place p_{12}. On the other hand, the internal subnet $\mathcal{N}_{1\ell}$ of $\mathcal{N}_1$ corresponds to a "loop" structure in the considered thread that requires lock r_3 for each iteration of its execution. Finally, notice that the depicted initial marking of the Gadara net of Fig. 1 obeys the requirements regarding the marking of the operation and the resource places that were discussed in the previous paragraph. Also, in the considered example, we have picked $M(p_{0i}) = 2,\ i = 1, 2$, which is the maximal number of process instances that can execute concurrently in $\mathcal{N}_1$ or $\mathcal{N}_2$.

B. Extended Finite Automata

An extended finite automaton (EFA) is an augmentation of the ordinary finite state automaton (FSA) with integer variables that are employed in a set of guards and maintained by a set of actions. A transition in an EFA is enabled if and

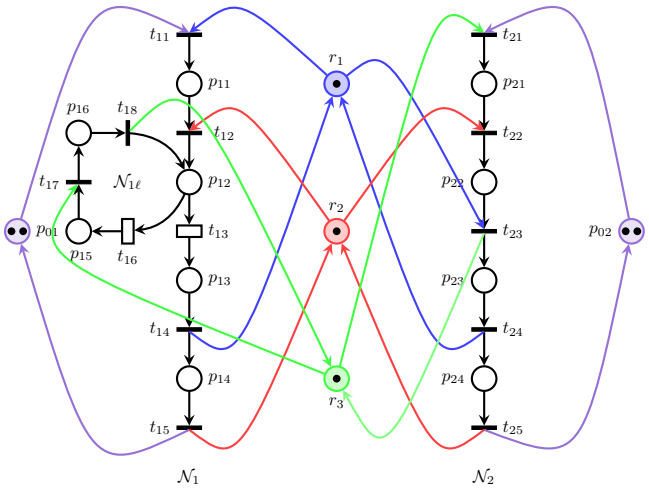


Fig. 1: A Gadara net model of two threads sharing three locks.

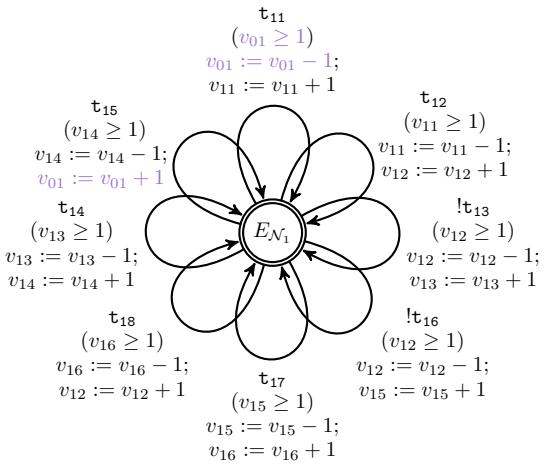


Fig. 2: The EFA modeling the process subnet $\mathcal{N}_1$.

only if its corresponding guard is true. Once a transition is taken, updating actions on the set of variables may follow. By utilizing these two mechanisms, an EFA can represent the modeled behavior in a conciser manner than the ordinary FSA model.

More formally, an EFA over a set of integer variables $v = (v_1, \ldots, v_n)$ is a 5-tuple $E = (Q, \Sigma, \rightarrow, s_0, Q^m)$, where (i) $Q : L \times \mathcal{D}$ is the extended finite set of states. L is the finite set of the model *locations* and $\mathcal{D} = \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ is the finite domain of the model *variables* $v = (v_1, \ldots, v_n)$. (ii) Σ is a nonempty finite set of *events* (a.k.a. the *alphabet* of the model). (iii) $\rightarrow \subseteq L \times \Sigma \times G \times A \times L$ is the *transition relation*, describing a set of transitions that take place among the model locations upon the occurrence of certain events. However, these transitions are further qualified by G, which is a set of *guard* predicates defined on $\mathcal{D}$, and by A, which is a collection of *actions* that update the model variables as a consequence of an occurring transition. Each action $a \in A$ is an n-tuple of functions $(a_1, \ldots, a_n)$, with each function a_i updating the corresponding variable v_i. (iv) $s_0 = (\ell_0, v_0) \in L \times \mathcal{D}$ is the *initial state*, where ℓ_0 is the *initial location*, while v_0 denotes the vector of the *initial values* for the model variables. (v) $Q^m \subseteq L^m \times \mathcal{D}^m \subseteq Q$ is the set of *marked states*. $L^m \subseteq L$ is the set of the *marked locations* and $\mathcal{D}^m \subseteq \mathcal{D}$ denotes the set of the vectors of *marked values* for the model variables. For the sake of brevity, in the following, we shall use the notation $\ell \xrightarrow{\sigma}_{g/a} \ell'$ as an abbreviation for $(\ell, \sigma, g, a, \ell') \in \rightarrow$.

C. Modeling Gadara Nets as EFA

To apply the symbolic approaches of [3], [4] for the computation of the maximally permissive DAP of a Gadara net, we need to re-cast this net into an equivalent EFA model. Next, we demonstrate this conversion through the Gadara net that is depicted in Fig. 1.

Declaration of the resource variables. We start with the declaration of a set of variables that represent the resource places in $\mathcal{N}_G$. For each place $r_i \in P_R$, where $i = 1, 2, 3$, we introduce a *resource variable* vr_i to trace the availability of the mutex lock that is represented by r_i. The domain of vr_i is $\{0, 1\}$. Furthermore, we define value 1 as the initial and the marked value of each variable vr_i, since, under proper execution, the initial and the marked state correspond to the marking M_0 of $\mathcal{N}_G$.

Representation of a process subnet by a single-location EFA. Next, we proceed to build the EFA that captures the execution of the thread that is modeled by the process subnet $\mathcal{N}_1$. At a first phase, we construct an EFA that concerns only the representation of the routing possibilities among the operation places of $\mathcal{N}_1$, and it does not address the relevant mutex lock allocation function; this function will be modeled in a subsequent phase.

As indicated in Fig. 2, the constructed EFA has only one location, and its eight transitions correspond to the transitions $t_{11}, \ldots, t_{18}$ of the process subnet $\mathcal{N}_1$ in Fig. 1. Notice that we prepend an exclamation mark (!) in the front of the transitions t_{13}, t_{16} to indicate that they are uncontrollable. Next, we define the set of *process variables* v_{1j}, $j = 1, \ldots, 6$, that count the number of tokens in the corresponding operation places p_{1j} of $\mathcal{N}_1$. Also, we define the *idle variable* v_{01} to count the number of tokens in the idle place p_{01}, with the initial and marked value set equal to 2. By making use of these instance-counting variables, we can construct the necessary guards and actions for the EFA transitions. As depicted in Fig. 2, these guards determine whether a transition can take place, on the basis of the token availability at the originating places. Upon the occurrence of such a transition, the corresponding actions update accordingly the number of the tokens at the various places.

Representation of the mutex lock allocation and its induced dynamics. Finally, we extend the EFA depicted in Fig. 1 with the resource variables vr_i to model the complete behavior of the process subnet $\mathcal{N}_1$. While doing this, we also perform several reductions on the resulting EFA to obtain a simpler, yet equally expressive model w.r.t. the task of deadlock avoidance. Regarding these reductions, first it can be observed from Fig. 1 that the transitions t_{14} and t_{15} of $\mathcal{N}_1$ do not pose any resource requests. Hence, if

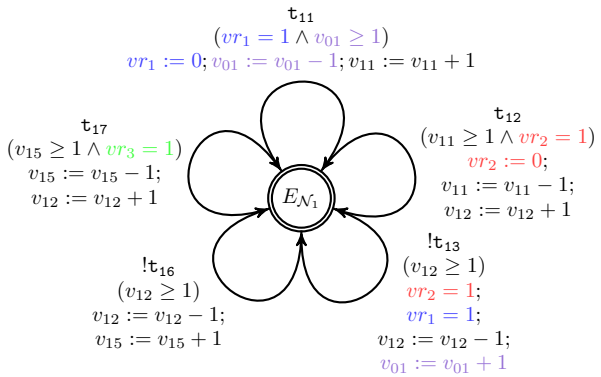


Fig. 3: The resource-augmented EFA for $\mathcal{N}_1$.

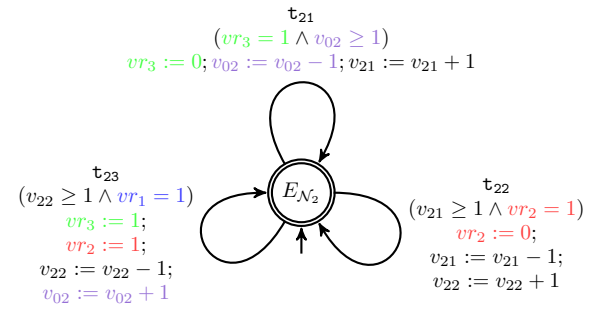


Fig. 4: The resource-augmented EFA for $\mathcal{N}_2$.

a token is present in place p_{13}, transitions t_{14} and t_{15} of $\mathcal{N}_1$ will be eventually executed without causing deadlocks. Therefore, we could conveniently forgo transition t_{15} of the EFA and append all its actions to those of transition t_{14}. After merging t_{14} with t_{15}, the variable v_{14} is insignificant, and, thus, its associated guards and actions can be omitted. A similar reasoning can lead to the merging of transition t_{13} with the merged transition t_{14} and even to the merging of transition t_{18} with transition t_{17} in the internal subset $\mathcal{N}_{1\ell}$; and, of course, the insignificant variables v_{13} and v_{16} and their associated actions and guards are accordingly omitted. As a result of all this transition merging, the final resource-augmented EFA for process subnet $\mathcal{N}_1$ has five transitions $t_{11}, t_{12}, t_{13}, t_{16}$ and t_{17} with the transitions t_{11}, t_{12}, t_{17} corresponding to the allocation of one lock and transitions t_{13} and t_{16} corresponding to the branching decisions at the operation place p_{12}; this EFA will be denoted by $E_{\mathcal{N}_1}$, and it is depicted in Fig. 3. The reader should also notice that the transition t_{17} in Fig. 3 has no action associated with the lock vr_3, since both the allocation and release operations of vr_3 are executed at the (merged) transition t_{17}. Finally, the EFA $E_{\mathcal{N}_2}$, that models the complete behavior of the process subnet $\mathcal{N}_2$, can be constructed in a similar manner; this EFA is depicted in Fig. 4.

State feasibility. In the representation of the EFA model, the place invariants of the Gadara net that are enforced by its three resource places, are expressed by the following three constraints for the model variables:

$$\begin{aligned} v_{11} + v_{12} + v_{15} + vr_1 &= 1; \\ v_{12} + v_{22} + vr_2 &= 1; \\ v_{21} + v_{22} + vr_3 &= 1. \end{aligned} \tag{1}$$

On the other hand, the strongly-connected-state-machine structure of each process subnet of the Gadara net of Fig. 1 implies an additional set of place invariants that are expressed by the following two equations in the EFA model:

$$v_{11} + v_{12} + v_{15} + v_{01} = 2;\ v_{21} + v_{22} + v_{02} = 2. \tag{2}$$

In the following, a state s of the (global) EFA consisting of $E_{\mathcal{N}_1}$ and $E_{\mathcal{N}_2}$, with a variable vector v satisfying the constraints of (1,2), will be characterized as a *feasible state*.

III. Computing the Minimal Boundary Unsafe States

In Section II, we have showed how the Gadara net that models a multithreaded program can be re-cast to an equivalent (set of) EFA, with each EFA in the derived set modeling a resource-augmented process subnet. In this section, we focus on the BDD-based symbolic computation of the minimal boundary unsafe states in the considered EFA model.

A. Binary Decision Diagrams

Binary decision diagrams (BDDs) [5] are a memory-efficient data structure used to represent Boolean functions. For any Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ in n Boolean variables $X = (x_1, \ldots, x_n)$, a BDD-based representation of f is a graphical representation of this function that is based on the following identity:

$$\forall x_i \in X,\ f = (\neg x_i \wedge f|_{x_i=0}) \vee (x_i \wedge f|_{x_i=1}), \tag{3}$$

where $f|_{x_i=0}$(resp. 1) denotes the Boolean function that is induced from function f by fixing the value of variable x_i to 0 (resp. 1). More specifically, (3) enables the representation of f as a single-rooted acyclic digraph with two types of nodes: *decision nodes* and *terminal nodes*. A terminal node can be labeled either 0 or 1. Each decision node is labelled by a Boolean variable and it has two outgoing edges, with each edge corresponding to assigning the value of the labeling variable to 0 or to 1. The value of function f is evaluated by starting from the root of the BDD and at each visited node following the edge that corresponds to the selected value for the node-labeling variable; the value of f is the value of the terminal node that is reached through this path.

The *size* of a BDD refers to the number of its decision nodes. A carefully structured BDD can provide a more compact representation for a Boolean function f than the corresponding truth table and the decision tree; frequently, the attained compression is by orders of magnitude.

From a computational standpoint, the efficiency of BDDs is mainly due to the fact that the worst-case complexity of performing some logical operation on two functions f and f' is $\mathcal{O}(|f| \cdot |f'|)$, where $|f|$ and $|f'|$ are the sizes of the BDDs representing f and f'.

EFA encoding through BDDs. For the encoding of the state set of an EFA, $Q : L \times \mathcal{D}$, we employ two Boolean

variable sets, denoted by X^L and $X^{\mathcal{D}} = X^{\mathcal{D}_1} \cup \ldots \cup X^{\mathcal{D}_n}$, to respectively encode the two sets L and $\mathcal{D}$. Then, each state $q = (\ell, v) \in Q$ is associated with a unique satisfying assignment of the variables in $X^L \cup X^{\mathcal{D}}$. Given a subset $\bar{Q}$ of Q, its *characteristic function* $\chi_{\bar{Q}} : Q \rightarrow \{0, 1\}$ assigns the value of 1 to all states $q \in \bar{Q}$ and the value of 0 to all states $q \notin \bar{Q}$.[2] The symbolic representation of the transition relation $\rightarrow$ relies on the same idea. A transition is essentially a tuple $\langle \ell, v, \sigma, \ell', v' \rangle$ specifying a source state $q = (\ell, v)$, an event σ, and a target state $q' = (\ell', v')$. Formally, we employ the variable sets X^L and $X^{\mathcal{D}}$ to encode the source state q, and a copy of X^L and $X^{\mathcal{D}}$, denoted by $\acute{X}^L$ and $\acute{X}^{\mathcal{D}}$, to encode the target state q'. In addition, we employ the Boolean variable set X^{Σ} to encode the alphabet of E, and we associate the event σ with a unique satisfying assignment of the variables in X^{Σ}. Then, we identify the transition relation $\rightarrow$ of E with the characteristic function

$$\Delta(\langle q, \sigma, q' \rangle) = \begin{cases} 1 & \text{if } (\ell, \sigma, g, a, \ell') \in \rightarrow, v \models g, v' = a(v) \\ 0 & \text{otherwise} \end{cases}$$

That is, Δ assigns value of 1 to $\langle q, \sigma, q' \rangle$ if there is a transition from ℓ to ℓ' labelled by σ, the values of variables at ℓ satisfy the guard g, i.e., $v \models g$, and the values of variables v' at ℓ' are the result of performing action a on v.

Having the distinct EFA $E_{\mathcal{N}_i}, i = 1, 2$, that model the process subnets $\mathcal{N}_1$ and $\mathcal{N}_2$ of the Gadara net $\mathcal{N}_G$ of Fig. 1, we shall denote by Δ_i the corresponding symbolic representations of $E_{\mathcal{N}_i}$. The global lock allocation and release dynamics generated by the considered $\mathcal{N}_G$ can be formally expressed by the extended full synchronous composition, introduced in [12], that composes the aforementioned EFA to the "plant" EFA $\mathbf{E} = E_{\mathcal{N}_1} || E_{\mathcal{N}_2}$. A symbolic representation of $\mathbf{E}$ will be denoted by $\Delta_{\mathbf{E}}$, and it can be systematically obtained from Δ_1, Δ_2 by using the approach introduced in [13]; the discussion of this approach is beyond the scope of this work, and, thus, we refer to [13] for the details.

Fig. 5 depicts the dynamic behavior of the composed EFA $\mathbf{E}$ using the BDD-related concepts that were discussed in the previous paragraph. The depicted state transition diagram (STD) includes only the feasible states that are reachable from the (composed) initial state $\mathbf{s_0}$, and furthermore, it considers only those states that are modeled explicitly in this EFA through the values of the corresponding variables. As it can be seen in Fig. 5, the resulting STD involves ten (10) states, with each state s_i, $i = 0, \ldots, 9$, being described by ten components that correspond to the values of the idle and process variables $v_{01}, v_{11}, v_{12}, v_{15}, v_{02}, v_{21}, v_{22}$ and the resource variables vr_1, vr_2, vr_3.

B. Computation of the Minimal Boundary Unsafe States

The BDD-based approach of [3] employs a three-stage computation for identifying the minimal boundary unsafe states from the symbolically represented transition relation of the composed EFA $\mathbf{E}$, i.e., from $\Delta_{\mathbf{E}}$. In the first stage, all the deadlock states are identified and retrieved from $\Delta_{\mathbf{E}}$. In the second stage, the deadlock states are used as starting points for a search procedure over $\Delta_{\mathbf{E}}$ that identifies all the boundary unsafe states. In the third stage, the minimal boundary unsafe states are extracted from the computed set of the boundary unsafe states. In general, the computed state sets may include some states that are not reachable from the initial state $\mathbf{s_0}$, but this does not impede the implementation of the maximally permissive DAP by means of these sets and the one-step-lookahead logic that was outlined in the earlier parts of this manuscript.

[2] In the rest of the paper, we shall use interchangeably the original name of a set Q and its characteristic function, χ_Q, in order to refer to this set.

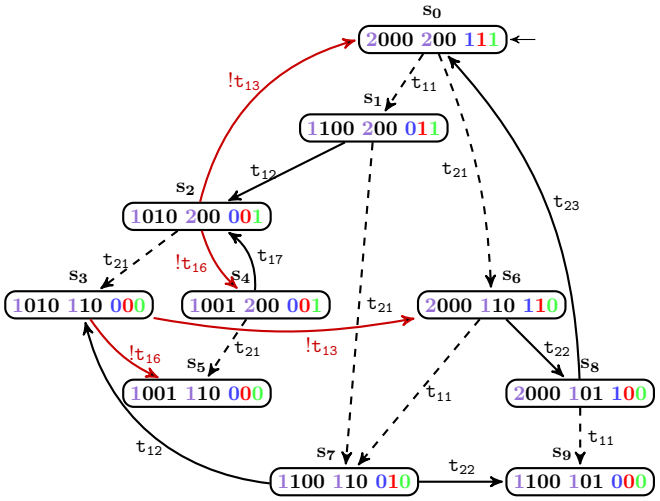


Fig. 5: The state transition diagram (STD) modeling the dynamic behavior of the EFA $\mathbf{E} = E_{\mathcal{N}_1} || E_{\mathcal{N}_2}$. In the depicted STD, the dashed arcs correspond to the (thread-instance) *loading* transitions to a critical region; the solid arcs correspond to the (thread-instance) *advancing* transitions within the corresponding critical regions, with the red arcs further corresponding to the *uncontrollable branching* transitions. In the approach presented in [3], this STD is symbolically represented by a BDD that also includes other unreachable and/or infeasible states.

The rest of this section illustrates the aforementioned computation on the STD in Fig. 5. For reasons that will become clear in the following, it is pertinent to assume that $\Delta_{\mathbf{E}}$ is partitioned into two BDDs Δ_L and Δ_A that collect respectively the transitions in $\Delta_{\mathbf{E}}$ corresponding to the loading transitions t_{11} and t_{21}, and the advancing transitions $t_{12}, t_{13}, t_{16}, t_{17}, t_{22}$ and t_{23}.

Identification of the feasible deadlock states. In the context of this work, a deadlock state is a state of the EFA $\mathbf{E}$ that contains some activated threads (i.e., it is different from the initial state) and it does not enable any thread-advancing transitions. With $\Delta_{\mathbf{E}}$ available, this set of states can be symbolically computed as follows: First, we collect all the "target" states of the transitions in $\Delta_{\mathbf{E}}$ and we denote the corresponding set as the set T. In the considered example, T will contain all the states $\mathbf{s_i}$, $\mathbf{i = 0, \ldots, 9}$, in the STD of Fig. 5, but it also contains other infeasible states that have resulted from the composition of the EFAs $E_{\mathcal{N}_1}$ and $E_{\mathcal{N}_2}$. Subsequently, we collect the set of all the "source" states of the transitions in Δ_A and we denote this set by E; in the considered example, $E = \{\mathbf{s_1}, \mathbf{s_2}, \mathbf{s_3}, \mathbf{s_4}, \mathbf{s_6}, \mathbf{s_7}, \mathbf{s_8}\}$. The reader should notice that state $\mathbf{s_0} \notin E$, since none of the

relevant transitions $\langle \mathbf{s_0}, t_{11}, \mathbf{s_1} \rangle$, $\langle \mathbf{s_0}, t_{21}, \mathbf{s_6} \rangle$ belongs to Δ_A. Finally, we (symbolically) compute the set of all the deadlock states, D, with respect to the thread advancing transitions, by removing from set T the initial state $\mathbf{s_0}$, and all the states belonging to the state set E.

Since set D is computed from the entire set of transitions that is contained in $\Delta_{\mathbf{E}}$, it might contain deadlock states that are infeasible (i.e., they violate the constraints of (1, 2)). The presence of these infeasible states in D would increase unnecessarily the computational cost of the second stage, that is discussed next. Hence, as the last step of this first stage, the symbolic representation of D, χ_D, is filtered through its conjunction with the BDD χ_F that encodes the constraints (1, 2), in order to obtain the set of feasible deadlock states; this set is denoted by FD. $FD = \{\mathbf{s_5}, \mathbf{s_9}\}$ for the STD depicted in Fig. 5.

Identification of the feasible boundary unsafe states. Having obtained the set FD of the feasible deadlock states, the algorithm of [3] proceeds with the symbolic computation of the feasible boundary unsafe state set, denoted by FB. The algorithm employs the set U to collect all the identified unsafe states. Also, at each iteration, the set U_{new} defines the set of the unsafe states that are to be processed at that iteration, through one-step-backtracking in $\Delta_{\mathbf{E}}$, in an effort to reach and explore new states. Both U and U_{new} are initialized to FD. Moreover, we define the transition set $\hat{U}_{pre} = \{(s, u) \in \Delta_A \mid u \in U \text{ and } s \notin U\}$; during the entire search process, $\hat{U}_{pre}$ contains the transitions of Δ_A where the target states belong to U while the source states have also transitions to states that currently are not in U. Initially, $\hat{U}_{pre}$ is empty.

We start with the extraction of all the states that can be reached from U_{new} by backtracing some transitions in Δ_A. We shall denote by $\Delta_{\hat{U}}$ the set of the relevant transitions. With respect to the STD depicted in Fig. 5, $\Delta_{\hat{U}} = \{\langle \mathbf{s_3}, t_{16}, \mathbf{s_5} \rangle, \langle \mathbf{s_7}, t_{22}, \mathbf{s_9} \rangle\}$. Among the transitions in $\Delta_{\hat{U}}$, notice that the source state $\mathbf{s_3}$ of the transition $\langle \mathbf{s_3}, t_{16}, \mathbf{s_5} \rangle$ reaches the unsafe state $\mathbf{s_5}$ uncontrollably due to the uncontrollable event (transition) t_{16}. Therefore, we can determine that state $\mathbf{s_3}$ is unsafe immediately. The identified unsafe state $\mathbf{s_3}$ is appended into the set U_{curr} that denotes the set of unsafe states identified at the current iteration. The transition set $\Delta_{\hat{U}}$ is then updated accordingly by removing the transitions with the sources in U_{curr}. We then collect the source state $\mathbf{s_7}$ of the remaining transition $\langle \mathbf{s_7}, t_{22}, \mathbf{s_9} \rangle$ in $\Delta_{\hat{U}}$ and store it into the state set $S\hat{U}$. Subsequently, we perform a one-step forward search over Δ_A starting from the elements in the state set $S\hat{U}$, and collect all the transitions of Δ_A that originate from some element in $S\hat{U}$; these transitions are stored in the set Δ_{SA}. In the considered example, $\Delta_{SA} = \{\langle \mathbf{s_7}, t_{12}, \mathbf{s_3} \rangle, \langle \mathbf{s_7}, t_{22}, \mathbf{s_9} \rangle\}$. By removing from Δ_{SA} all the transitions belonging to $\Delta_{\hat{U}}$ and $\hat{U}_{pre}$, and extracting the source states of the remaining transitions, we can identify the states in $S\hat{U}$ that are not recognized as unsafe states at the current iteration. Subtracting this last set of states from $S\hat{U}$, leaves a set of unsafe states that are appended to U_{curr}. In the context of the considered example, state s_7 (which is the unique element of $S\hat{U}$) cannot be classified as an unsafe state by the aforementioned computations during the current iteration. At the end of the iteration, all the unsafe states in U_{curr} that are already in U are removed from this set. The remaining states in U_{curr} are appended in U, and they also re-initialize the set U_{new}. Hence, in the considered example, $U_{new} = \{\mathbf{s_3}\}$. Also, the transition $\langle \mathbf{s_7}, t_{22}, \mathbf{s_9} \rangle$ is added to the set $\hat{U}_{pre}$.

Following the logic described in the previous paragraphs, in the second iteration of the considered search process, state $\mathbf{s_3}$ is used to backtrace over Δ_A, and the transition $\langle \mathbf{s_7}, t_{12}, \mathbf{s_3} \rangle$ is identified and stored into set $\Delta_{\hat{U}}$. The source state $\mathbf{s_7}$ of the transition is then used to perform the one-step reachability over Δ_A, and $\Delta_{SA} = \{\langle \mathbf{s_7}, t_{12}, \mathbf{s_3} \rangle, \langle \mathbf{s_7}, t_{22}, \mathbf{s_9} \rangle\}$. Since both of these transitions belong in $\Delta_{\hat{U}} \cup \hat{U}_{pre}$, state $\mathbf{s_7}$ is identified as an unsafe state in this iteration. Consequently, U is updated to $\{\mathbf{s_3}, \mathbf{s_5}, \mathbf{s_7}, \mathbf{s_9}\}$ and U_{new} is set to $\{\mathbf{s_7}\}$.

The backward search process terminates after the third iteration, since no new unsafe state can be identified when backtracing from state $\mathbf{s_7}$ over Δ_A. At this point, the symbolic approach proceeds to extract the boundary states from the overall set of unsafe states, U. For that, the symbolic computation extracts from $\Delta_{\mathbf{E}}$ all the transitions with their target states belonging to the states in U; the relevant transition set is denoted by $\Delta_{\mathcal{B}}$. Next, the algorithm retrieves from $\Delta_{\mathcal{B}}$ the transition set $\Delta_{S\mathcal{B}}$, where the source states of the included transitions are safe states. Finally, the boundary unsafe state set FB is obtained by extracting the target states from $\Delta_{S\mathcal{B}}$. In the context of the STD in Fig. 5, the execution of the aforementioned operations results in $FB = U$; i.e., all the unsafe states in the previously computed set U are boundary unsafe states.

Identification of the minimal boundary unsafe states. An important implication of the invariants of (1, 2) is that, at any feasible state of the underlying EFA state-space, the values of the resource variables and the idle variables can be induced from the values of the process variables. In other words, any feasible state s of the considered STD can be uniquely determined only by the specification of its process variables. Hence, one can obtain a more compact symbolic representation of the set of feasible boundary unsafe states, χ_{FB}, by eliminating from the elements of χ_{FB} the values that correspond to the idle and resource variables. Letting X^R and X^I respectively denote the Boolean variables representing the values of the resource variables vr_i, $i = 1, 2, 3$, and the idle variables v_{01} and v_{02}, this elimination can be performed through the following existential quantification:

$$\chi_{FB} := \exists (X^R \cup X^I).\ \chi_{FB}. \tag{4}$$

For the considered example, the state set FB that is returned by the operation of Eq. 4 can be represented as follows: $FB = \{01010(\mathbf{s_3}), 00110(\mathbf{s_5}), 10010(\mathbf{s_7}), 10001(\mathbf{s_9})\}$ (i.e., the boundary unsafe states are described only by their process variables).

Given any two feasible boundary unsafe states $\mathbf{s}$, $\mathbf{s}'$ represented according to the logic of (4), we consider the ordering relation "$\leq$" on them that is defined by the application of this relation componentwise; i.e.,

$$\mathbf{s} \leq \mathbf{s}' \iff (\forall k = 1, \ldots, K, \mathbf{s}[k] \leq \mathbf{s}'[k]), \tag{5}$$

where $\mathbf{s}[k]$ and $\mathbf{s}'[k]$ are the values of the k-th process variable for $\mathbf{s}$ and $\mathbf{s}'$. Furthermore, we use the notation '$<$' to denote that condition (5) holds as strict inequality for at least one component $v_k \in \{v_1, \ldots, v_K\}$. It is shown in [6] that if state $\mathbf{s}$ is unsafe and state $\mathbf{s}'$ satisfies $\mathbf{s} \leq \mathbf{s}'$, then the state $\mathbf{s}'$ is also unsafe. Hence, under the state representation of (4), the set FB can be effectively defined by the subset of its minimal elements. We shall denote this subset by $\overline{FB}$, i.e., $\overline{FB} \equiv \{\mathbf{s} \in FB \mid \nexists \mathbf{s}' \in FB \text{ s.t. } \mathbf{s}' < \mathbf{s}\}$. A symbolic algorithm for the computation of $\overline{FB}$ from FB is provided in [3]. We also notice, for completeness, that in the considered example, $\overline{FB} = FB$.

IV. Representing the Target DAP as Guards

As remarked in the introductory section, the availability of the set of boundary minimal unsafe states, $\overline{FB}$, enables the implementation of the maximally permissive DAP for the considered RAS through the one-step-lookahead control scheme that identifies and blocks transitions to states that dominate some element of $\overline{FB}$. In this section we develop a predicate g on the process variables of the underlying EFA $\mathbf{E}$ that will render this test more efficient. From an operational standpoint, the derived predicate g can be employed as an extra "guard" for those transitions t that can take the EFA $\mathbf{E}$ from its safe to its unsafe region. We remind the reader that this set of transitions is obtained as a "byproduct" of the computation that was described in the previous section; more specifically, these transitions are the transitions appearing in the elements of the derived set $\Delta_{S\mathcal{B}}$ in that computation. In the context of the considered example,

$$\Delta_{S\mathcal{B}} = \{\langle \mathbf{s_6}, t_{11}, \mathbf{s_7}\rangle, \langle \mathbf{s_8}, t_{11}, \mathbf{s_9}\rangle, \langle \mathbf{s_1}, t_{21}, \mathbf{s_7}\rangle, \langle \mathbf{s_2}, t_{21}, \mathbf{s_3}\rangle, \langle \mathbf{s_4}, t_{21}, \mathbf{s_5}\rangle\}.$$

Hence, the transitions that need further guarding are the transitions t_{11} and t_{21}.

With the set $\Delta_{S\mathcal{B}}$ readily available, the symbolic computation of the sought predicate g proceeds in the following steps: (i) First, we convert the content of the BDD representing the state set $\overline{FB}$ into an *integer decision diagram* (IDD) [14]. (ii) Next, we use the IDD derived in Step (i) in order to develop a predicate φ, defined on the process variables of the EFA $\mathbf{E}$, that recognizes all the states that are greater than or equal (component-wise) to some element of $\overline{FB}$, i.e., the states that "dominate" some element in $\overline{FB}$. (iii) Finally, the sought predicate g is obtained by setting $g := \neg\varphi$. Next, we elaborate further on the first two steps outlined above.

IDD generation for the state set $\overline{FB}$. An IDD is a generalization of the BDD concept where the number of terminals can be arbitrary and the domain of the variables that label each of the internal nodes in the diagram can be an arbitrary set of integers. In this paper, we use IDDs with only one terminal, the 1-terminal. A practical value of IDDs is that they can provide an explicit, and yet very compact, representation to any set of integer vectors by taking advantage of the commonality that might exist in various segments of these vectors.[3] Furthermore, in the following we shall show that IDDs can be re-hashed very straightforwardly into predicates that provide an alternative representation of their information (state vector) content. In the context of the considered application of Gadara nets, this conversion process is very efficient and the resulting predicates are pretty compact.

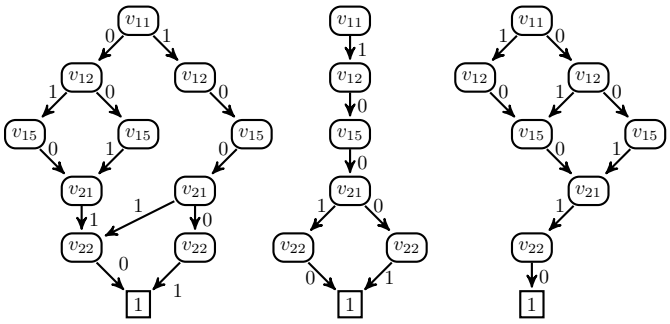
An algorithm for converting a BDD containing a set of EFA states to an IDD is provided in [7]. This algorithm assumes that the BDD binary variables are ordered in a way that observes the grouping and the sequencing of these variables that are established by the binary representation of the primary variables employed in the underlying EFA model. Under this condition, the algorithm traverses iteratively the provided BDD in a top-down depth-first manner, "segmenting" the traversed paths into sequences of integer values corresponding to the various EFA model variables. At the same time, these sequences are organized into the graph structure that eventually defines the computed IDD. We refer the reader to [7] for the computational details. The IDD that is obtained by the application of the algorithm of [7] on the BDD representing the state set $\overline{FB}$ is depicted in Fig. 6a.

Closing the discussion on the IDD development, we should further notice that, since (i) under the state representation of Eq. 4, the considered state sets are expressed only through the process variables of the underlying EFA, and (ii) each of these process variables is of binary nature (due to binary nature of the locks that are employed by the corresponding process stage in the Gadara net), the vectors that will be contained in the constructed IDD are still of a binary nature. This remark simplifies considerably the IDD construction process that was outlined in the previous paragraph, and it has additional significant implications for the predicate structure and the predicate generation process that are discussed in the remaining part of this section.

Developing the predicate φ from the derived IDD. The generation of the predicate φ that recognizes the states dominating some element in $\overline{FB}$, can be obtained from the IDD constructed in the previous paragraphs through a depth-first traversal of this graph that composes predicate φ according to the following "labeling" scheme: (i) An arc $a = (n, n')$ emanating from a node n that is labeled by the process variable v_{ij} of the considered IDD, and corresponding to some value k for this variable, is marked with the logical condition $\varphi(a) := v_{ij} \geq k$, if the node n' of this arc is the terminal node of the IDD. If node n' is non-terminal, then $\varphi(a) := (v_{ij} \geq k) \wedge \varphi(n')$, where $\varphi(n')$ is the logical condition that marks node n' (defined next). (ii) An internal node n of the considered IDD is marked with the logical condition $\varphi(n) := \bigvee_{a \in \mathcal{A}(n)} \varphi(a)$, where $\mathcal{A}(n)$ denotes the set of arcs a that emanate from node n. At the end, $\varphi := \varphi(n_0)$, where n_0 is the "source" node of the considered IDD.

Furthermore, taking into consideration the binary nature of the process variables v_{ij} in the context of the considered application, any generated condition $(v_{ij} \geq 0)$ in the above derivation can be treated as a tautology, while the condition $(v_{ij} \geq 1)$ can be expressed more compactly by v_{ij}. The application of the aforementioned logic to the IDD of Figure 6a leads to the following predicate φ: $(v_{12} + v_{15}) \cdot v_{21} + v_{11} \cdot (v_{21} + v_{22})$.

[3] In this capacity, IDDs are very similar to the TRIE data structure [15] that was used in [16] for the efficient storage of the RAS minimal unsafe states; these states were extracted in [16] through more conventional representations and techniques employed by Supervisory Control Theory [17].



(a) IDD for $\overline{FB}$ (b) IDD for $\overline{FB}_{t_{11}}$ (c) IDD for $\overline{FB}_{t_{12}}$

Fig. 6: IDDs representing the state set $\overline{FB}$ and its two subsets $\overline{FB}_{t_{11}}$ and $\overline{FB}_{t_{21}}$, for the considered STD of Fig. 5.

Finally, as explained in the beginning of this section, the negation of φ defines the "guard" predicate that must be enforced upon every transition that appears in the transition set $\Delta_{S\mathcal{B}}$.

An alternative, more distributed implementation of the target DAP. It is evident from the above discussion that the derived predicate g that expresses the target DAP logic is uniform across all the guarded transitions, i.e., the transitions appearing in the set $\Delta_{S\mathcal{B}}$. Next, we outline briefly an alternative organization of the presented computation that can lead to more customized, and therefore, more compact predicates, g_t, for the transitions t that appear in the set $\Delta_{S\mathcal{B}}$. The defining idea of this scheme is to employ the information that is provided by the sets FB and $\Delta_{S\mathcal{B}}$ obtained from the original computation of Section III, in order to compute for each transition t in $\Delta_{S\mathcal{B}}$, the set FB_t of the boundary unsafe states reached through that transition. Each of these sets can be subsequently processed in the same way that was used for the monolithic set FB; i.e., for each t appearing in $\Delta_{S\mathcal{B}}$, we can first extract the corresponding set $\overline{FB}_t$ containing the minimal elements of FB_t, translate this set to an IDD, and then use this IDD to obtain the predicate φ_t that expresses the states dominating the elements of FB_t. Eventually, each transition t appearing in $\Delta_{S\mathcal{B}}$ will be guarded by the additional predicate $g_t := \neg\varphi_t$.

Execution of this plan on the considered example has led to the $\overline{FB}_t$ sets, for the two transitions t_{11} and t_{21} appearing in the set $\Delta_{S\mathcal{B}}$, that are represented by the IDDs of Figs 6b and 6c. The respective "guard" predicates are: $g_{t_{11}} = \neg(v_{11} \cdot (v_{21} + v_{22}))$ and $g_{t_{21}} = \neg((v_{11} + v_{12} + v_{15}) \cdot v_{21})$.

V. Conclusion

This paper has revisited the symbolic framework that was recently developed for the computation of the maximally permissive DAP of complex RAS, and it has augmented this framework with an additional capability that enables the implementation of the derived policy through a set of predicates on the (process) variables of the underlying EFA. The presented results were also customized and applied, through a highlighting example, to a problem of deadlock avoidance in multithreaded software. In this application context, the predicates derived by the proposed methodology enable a straightforward implementation of the target policy logic through the introduction and the proper updating of the EFA process variables v_{ij} in the underlying source code. At the same time, extensive experimentation reported in [3], [4] reveals that the presented methodology can handle very complex RAS structures with extremely large state spaces (in the order of billions of states). Hence, the presented framework holds a strong potential for providing robust, practical and efficient solutions to the deadlock avoidance and liveness-enforcing supervision problems that are experienced in the considered application domain. Furthermore, the presented results extend pretty naturally to many other application areas that have been addressed by the relevant literature.

References

[1] S. A. Reveliotis, *Real-time Management of Resource Allocation Systems: A Discrete Event Systems Approach*. NY, NY: Springer, 2005.

[2] M. Zhou and M. P. Fanti (editors), *Deadlock Resolution in Computer-Integrated Systems*. Singapore: Marcel Dekker, Inc., 2004.

[3] Z. Fei, S. Reveliotis, S. Miremadi, and K. Åkesson, "A BDD-based approach for designing maximally permissive deadlock avoidance policies for complex resource allocation systems," Chalmers University, Tech. Rep., 2013, http://publications.lib.chalmers.se/records/fulltext/186774/local_186774.pdf.

[4] Z. Fei, S. Reveliotis, and K. Åkesson, "Symbolic computation of boundary unsafe states in complex resource allocation systems using partitioning techniques," Chalmers University, Tech. Rep., 2014.

[5] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, 1992.

[6] S. Reveliotis and A. Nazeem, "Deadlock avoidance policies for automated manufacturing systems using finite state automata," in *Formal Methods in Manufacturing*, J. Campos, C. Seatzu, and X. Xie, Eds. CRC Press / Taylor and Francis, 2014, pp. 169–195.

[7] S. Miremadi, K. Åkesson, and B. Lennartson, "Symbolic computation of reduced guards in supervisory control," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 754–765, 2011.

[8] T. Kelly, Y. Wang, S. Lafortune, and S. Mahlke, "Eliminating concurrency bugs with control engineering," *Computer*, vol. 42, no. 12, pp. 52–60, Dec 2009.

[9] Y. Wang, H. Liao, S. Reveliotis, T. Kelly, S. Mahlke, and S. Lafortune, "Modeling and analysis of a special class of Petri nets arising in multithreaded programs," in *CDC 2009*, 2009.

[10] H. Liao, Y. Wang, H. K. Cho, J. Stanley, T. Kelly, S. Lafortune, S. Mahlke, and S. Reveliotis, "Concurrency bugs in multithreaded software: Modeling and analysis using Petri nets," *Discrete Event Systems: Theory and Applications*, vol. 23, pp. 157–195, 2013.

[11] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, 1989.

[12] M. Sköldstam, K. Åkesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," *Decision and Control, 2007 46th IEEE Conference on*, pp. 3387–3392, 2007.

[13] S. Miremadi, B. Lennartson, and K. Åkesson, "A BDD-based approach for modeling plant and supervisor by extended finite automata," *IEEE Transactions on Control Systems Technology*, vol. 20, no. 6, pp. 1421–1435, 2012.

[14] J. Gunnarsson, "Symbolic methods and tools for discrete event dynamic systems," Ph.D. dissertation, Electrical Engineering, Linköping University, Linköping, Sweden, 1997.

[15] P. Brass, *Advanced Data Structures*. NY,NY: Cambridge University Press, 2008.

[16] A. Nazeem and S. Reveliotis, "A practical approach for maximally permissive liveness-enforcing supervision of complex resource allocation systems," *IEEE Trans. on Automation Science and Engineering*, vol. 8, pp. 766–779, 2011.

[17] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems (2nd ed.)*. NY,NY: Springer, 2008.