A BDD-Based Approach for Designing Maximally Permissive Deadlock Avoidance Policies for Complex Resource Allocation Systems

Zhennan Fei, Spyros Reveliotis, *Senior Member, IEEE*, Sajed Miremadi, Knut Åkesson, *Member, IEEE*

Abstract—**In order to develop a computationally efficient implementation of the maximally permissive deadlock avoidance policy (DAP) for complex resource allocation systems (RAS), a recent approach focuses on the identification of a set of critical states of the underlying RAS state-space, referred to as minimal boundary unsafe states. The availability of this information enables an expedient one-step-lookahead scheme that prevents the RAS from reaching outside its safe region. The work presented in this paper seeks to develop a symbolic approach, based on binary decision diagrams (BDDs), for efficiently retrieving the (minimal) boundary unsafe states from the underlying RAS state-space. The presented results clearly demonstrate that symbolic computation enables the deployment of the maximally permissive DAP for complex RAS with very large structure and state-spaces with limited time and memory requirements. Furthermore, the involved computational costs are substantially reduced through the pertinent exploitation of the special structure that exists in the considered problem.**

Note to Practitioners – **A key component of the real-time control of many flexibly automated operations is the management of the allocation of a finite set of reusable resources among a set of concurrently executing processes so that this allocation remains deadlock-free. The corresponding problem is known as deadlock avoidance, and its resolution in a way that retains the sought operational flexibilities has been a challenging problem due to (i) the inability to easily foresee the longer-term implications of an imminent allocation and (ii) the very large sizes of the relevant state spaces that prevent an on-line assessment of these implications through exhaustive enumeration. A recent methodology has sought to address these complications through the off-line identification and storage of a set of critical states in the underlying state space that renders efficient the safety assessment of any given resource allocation. The results presented in this paper further extend and strengthen this methodology by complementing it with techniques borrowed from the area of symbolic computation; these techniques enable a more compressed representation of the underlying state spaces and of the various subsets and operations that are involved in the pursued computation.**

Index Terms—**Resource Allocation Systems, Discrete Event Systems, Deadlock Avoidance, Maximal Permissiveness, Supervisory Control Theory, Binary Decision Diagrams.**

I. Introduction

DEADLOCK avoidance for sequential, complex resource allocation systems (RAS) is a well-established problem in the *discrete event systems* (DES) literature [1], [2]. In its basic positioning, this problem concerns the coordinated allocation of the system resources to a set of 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 process types, the applied control policy avoids the visitation of RAS states from which deadlock is inevitable. From an application standpoint, the need for deadlock avoidance arises in many contemporary technological systems, including the material flow control of flexibly automated production systems [3], [4], [5], the traffic management of unmanned discrete material handling systems [6], [7], [8], the traffic control of railway and urban monorail transport systems [9], and the lock allocation that takes place among the various threads of parallelized computer programs [10], [11].

Preferably, deadlock avoidance should be carried out in the *maximally permissive* manner. The computation of the maximally permissive deadlock avoidance policy (DAP) for any given RAS can be based, in principle, on standard synthesis procedures borrowed from DES *supervisory control theory* (SCT) [12], [13]. These procedures express the underlying resource allocation dynamics as a finite state automaton (FSA), and subsequently they "trim" this automaton with respect to (w.r.t.) the state where the underlying RAS is depleted of any processes; i.e., this empty state defines the initial as well as the target state for any successful operational cycle of the considered RAS.

Yet, although the SCT framework provides a rigorous base for modeling, analyzing and eventually controlling the RAS dynamics, the computational complexity for the synthesis of the maximally permissive DAP is an NP-hard task for the majority of RAS behavior [1], [14]. Hence, significant effort has been expended over the past years to provide DAPs that are computationally tractable and remain efficient w.r.t. the criterion of maximal permissiveness. In many cases, this effort has been facilitated by the adoption of additional modeling formalisms that connect more explicitly the representation of the system behavior to the underlying system structure, and they are, thus, more compact and more amenable to processing during the synthesis phases. *Petri Nets* (PNs) [15] have been a particularly popular modeling framework in the aforementioned line of research, while some representative works of this research line are those presented in [3], [5], [16], [17] and [18]. Another line of research has sought to combine the representational and computational strengths of

Z. Fei, K. Åkesson and S. Miremadi are with the Automation Research Group, Department of Signals and Systems, Chalmers University of Technology, SE-412 96, Gothenburg, Sweden, e-mail: zhennan@chalmers.se.

S. Reveliotis is with the School of Industrial & Systems Engineering, Georgia Institute of Technology, e-mail: spyros@isye.gatech.edu.

the existing modeling frameworks in a synergistic manner. In [19], the authors compute the maximally permissive DAP in the FSA modeling framework, and subsequently they seek to translate the outcome of this computation in the PN modeling framework using concepts and tools from the theory of regions [20]. Such an approach is frequently limited, however, by the large size of the resulting PNs, and, also, by the potential inability of the PN framework to provide an effective representation of the target policy. Hence, more recently, works like those presented in [21], [22], [23] and [24] have sought to represent the maximally permissive DAP, that is originally computed in the FSA modeling framework, through other representations that might be more parsimonious than the PNs generated by the theory of regions, and more capable to encode the maximally permissive DAP across the entire spectrum of the considered RAS. In [24], which is one of the primary inspirations for the work presented in this paper, an FSA-based approach was presented where the deployment of the maximally permissive DAP is based on the identification and the efficient storage of a set of critical states, referred to as the *minimal boundary unsafe states* of the underlying state-space. These critical states define the boundary between the safe and unsafe subspaces, where, in the considered problem context, state safety is defined as co-accessibility w.r.t. the empty RAS state. A tentative transition is considered to be unsafe if the resulting state is greater than or equal, component-wise, to one of the minimal boundary unsafe states. Furthermore, the results of [25] complement the work of [24] by introducing an algorithm that enumerates all the minimal unsafe states while avoiding the complete enumeration of the RAS state-space. More specifically, the algorithm of [25] retrieves the minimal unsafe states through a localized computation that starts from the minimal RAS deadlocks and backtraces the RAS dynamics until it has retrieved all the minimal unsafe states lying on the boundary between the safe and unsafe subspaces.

While the approaches discussed above take advantage of the structure and particular properties of the underlying RAS, a different line of work has sought to develop the maximally permissive DAP through a symbolic computation that uses the concept of the *binary decision diagram* (BDD) [26], [27]. A BDD is an efficient data structure, which, under the right conditions, can reach logarithmic compression of the involved state-spaces [27]. However, the effective deployment of BDD-based representations in supervisory control is still a non-trivial task. Some specific endeavors made towards this direction are presented in [28], [29], [30], [31], [32] and [33]. In [33], an efficient BDD-based approach was developed for modeling and controlling general DES represented in the modeling framework of the *Extended Finite Automata* (EFA) [34]. An EFA is an ordinary FSA extended with integer variables. The richer structure and semantics that are provided by these variables enable the representation of the modeled behavior in a conciser manner than the ordinary FSA. Hence, the approach in [33] employs the EFA model for the initial representation of the plant behavior and the control specifications, and subsequently it uses BDDs for the computation and representation of the underlying state-space. Finally, control synthesis is carried out according to the standard perspectives provided by the classical SCT [12], [13], but in the symbolic context of the BDD-based representation.

Motivated by the above approaches and remarks, in this paper, we propose a BDD-based approach for the efficient development of the maximally permissive DAP of the considered RAS. In particular, we first show how the considered RAS can be recast into a compact EFA model without losing any information necessary for solving the deadlock avoidance problem. Secondly, we present a series of symbolic algorithms for computing, from the BDD-based representation of the underlying state-space, the set of the boundary unsafe states, and also the subset of this last set that contains its minimal elements. The BDD containing the (minimal) boundary unsafe states, that results from the aforementioned computations, eventually can be converted through the algorithm presented in [35] into an *integer decision diagram* (IDD), also known as the TRIE data structure, similar to that employed in [24].[1] Hence, eventually, the maximally permissive DAP can be implemented through an one-step-lookahead control scheme similar to that deployed in [24]; we refer to that work for implementational details, and for a more extensive discussion on the TRIE data structure and the efficiencies that it provides to the deployment of the control function that is considered in this work.

The presented developments provide two different algorithms for computing the boundary unsafe states between the safe and unsafe subspaces. The first algorithm is an extension of that presented in [33], and it involves a reachability and co-reachability analysis of the underlying state-space, in order to compute the trim of the underlying FSA. Once the safe states have been computed, a reachability analysis is carried out one more time in order to identify and extract the boundary unsafe states. The second algorithm leverages the structural and the computational perspectives regarding the RAS (un-)safety developed in [24] and [25]. More specifically, the algorithm employs the following two-stage computation: In the first stage, all the deadlock states are identified and computed from the symbolically represented state-space. In the second stage, the deadlock states are used as starting points for a search procedure over the RAS state-space that identifies all the boundary unsafe states. Finally, each of the aforementioned approaches can be coupled with a third symbolic algorithm that extracts the minimal elements from the computed set of the boundary unsafe states. The computational results that are reported in Section V indicate that the TRIE data structure that is necessary for the effective encoding of this last state set is considerably smaller, in terms of the employed number of nodes, than the TRIE data structure that encodes the entire set of the boundary unsafe states, and therefore, it can support a more parsimonious implementation of the maximally permissive DAP. The proposed symbolic approaches have been implemented and integrated into Supremica [36], a tool for analysis of DES. Our experimental results also reveal that the proposed symbolic computation enables the deployment of

[1]But, as already explained, the informational content of the TRIE data structure that is employed in [24], is obtained through enumerative techniques that employ a more conventional representation of the underlying RAS dynamics.

the maximally permissive DAP for complex RAS with very large structure and state-spaces, with limited time and memory requirements. Furthermore, the involved computational costs can be substantially reduced through the pertinent exploitation of the special structure that exists in the considered problem.

The rest of the paper is organized as follows: Section II provides a brief introduction to the class of the resource allocation systems that are considered in this work, and the corresponding problem of maximally permissive deadlock avoidance. Section III provides first a brief introduction of the EFA model, and subsequently discusses the employment of this model for the effective, formal representation of the RAS class that was introduced in Section II. Section IV introduces a symbolic representation for the EFA model that is developed in Section III, by means of BDDs, and leverages this representation towards the development of the symbolic algorithms for computing the target sets of states that were described in the previous paragraph. Section V presents the experimental results that demonstrate and assess the efficacy of the proposed algorithms, and, finally, Section VI concludes the paper by summarizing the contributions and outlining some future work. Furthermore, due to space considerations, some material supporting the presented developments but not deemed as absolutely necessary for a solid understanding of the paper results has been organized into an electronic supplement that is provided at [37].[2] Closing the discussion of this introductory section, we also notice, for completeness, that preliminary, abridged versions of some parts of the results that are provided in the manuscript were presented in [39], [40].

II. Resource Allocation Systems and the corresponding problem of Deadlock Avoidance

We start our technical developments by providing a formal characterization of the RAS class that is considered in this work.

Definition II.1: For the purposes of this work, a *resource allocation system (RAS)* is a 4-tuple $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ where:

- $\mathcal{R} = \{R_1, \ldots, R_m\}$ is the set of the system *resource types*.
- $C : \mathcal{R} \rightarrow \mathbb{Z}^+$ – where $\mathbb{Z}^+$ is the set of strictly positive integers – is the system *capacity* function, characterizing the number of identical units from each resource type available in the system. Resources are assumed to be *reusable*, i.e., each allocation cycle does not affect their functional status or subsequent availability, and therefore, $C(R_i) \equiv C_i$ constitutes a system *invariant* for each R_i.
- $\mathcal{P} = \{J_1, \ldots, J_n\}$ denotes the set of the system *process types* supported by the considered system configuration. Each process type J_j, for $j = 1, \ldots, n$, is a composite element itself; in particular, $J_j = \langle \mathcal{S}_j, \mathcal{G}_j \rangle$, where $\mathcal{S}_j = \{\Xi_{j1}, \ldots, \Xi_{j,l(j)}\}$ denotes the set of *processing stages* involved in the definition of process type J_j, and $\mathcal{G}_j$ is an *acyclic digraph* that defines the sequential logic of process type J_j. The node set of $\mathcal{G}_j$ is in one-to-one correspondence with the processing-stage set $\mathcal{S}_j$, and each directed path from a source node to a terminal node of $\mathcal{G}_j$ corresponds to a possible execution sequence (or "process plan") for process type J_j. Also, given an edge $e \in \mathcal{G}_j$ linking Ξ_{jk} to $\Xi_{jk'}$, we define $e.src \equiv \Xi_{jk}$ and $e.dst \equiv \Xi_{jk'}$, i.e., $e.src$ and $e.dst$ denote respectively the source and the destination nodes of edge e.
- $\mathcal{A} : \bigcup_{j=1}^{n} \mathcal{S}_j \rightarrow \prod_{i=1}^{m} \{0, \ldots, C_i\}$ is the *resource allocation function*, which associates every processing stage Ξ_{jk} with the *resource allocation request* $\mathcal{A}(j,k) \equiv \mathcal{A}_{jk}$. More specifically, each $\mathcal{A}(j,k)$ is an m-dimensional vector, with its i-th component indicating the number of resource units of resource type R_i necessary to support the execution of stage Ξ_{jk}. Furthermore, it is assumed that $\mathcal{A}_{jk} \neq \mathbf{0}$, i.e., every processing stage requires at least one resource unit for its execution. Finally, according to the applying resource allocation protocol, a process instance executing a processing stage Ξ_{jk} will be able to advance to a successor processing stage $\Xi_{jk'}$, only after it is allocated the resource differential $(\mathcal{A}_{jk'} - \mathcal{A}_{jk})^+$; and it is only upon this advancement that the process will release the resource units $|(\mathcal{A}_{jk'} - \mathcal{A}_{jk})^-|$, that are not needed anymore.[3]

The "hold-while-waiting" protocol that is described in the last part of Definition II.1, when combined with the arbitrary nature of the process routes and the resource allocation requests that are supported by the considered RAS model, can give rise to resource allocation states where a set of processes are waiting upon each other for the release of resources that are necessary for their advancement to their next processing stage. Such persisting cyclical-waiting patterns are known as *(partial) deadlocks* in the relevant literature, and to the extent that they disrupt the smooth operation of the underlying system, they must be recognized and eliminated from the system behavior. The relevant control problem is known as *deadlock avoidance*, and as remarked in the introductory section, a natural framework for its investigation is that of SCT [12], [13]. More specifically, in an FSA-based representation of the RAS dynamics, deadlocks appear as states containing a set of activated process instances and no feasible process-advancing events. Hence, assuming that the desired outcome of any run of this FSA is the access of the state where all processes have successfully completed and the underlying RAS is idle and empty of any active processes, the presence of deadlock states can be perceived as *blocking* behavior. Therefore, in the context of SCT, effective deadlock avoidance translates to the development of the *maximally permissive non-blocking supervisor* for the RAS-modeling FSA, that will confine the RAS behavior in the "trim" of this FSA, i.e., to the subspace consisting of the states that are reachable and co-reachable to the RAS idle and empty state.

In the relevant RAS theory, states that are co-reachable to the RAS idle and empty state are also characterized as *safe*, and, correspondingly, states that are not co-reachable are characterized as *unsafe*. Furthermore, it is evident from the above discussion that of particular interest in the implementation of the maximally permissive non-blocking supervisor for the considered RAS are those transitions leading from safe to

[2]An alternative source of this supplementary material, that also provides additional context for this entire research, is [38].

[3]We remind the reader that $(x)^+ = \max\{0, x\}$ and $(x)^- = \min\{0, x\}$.

unsafe states, since their effective recognition and blockage can prevent entrance into the unsafe region. The unsafe states that result from such problematic transitions are known as the *boundary unsafe* states in the relevant literature. Also, for reasons that will be explained in the sequel, the entire set of the boundary unsafe states can be effectively recognized from its minimal elements. *Hence, the main subject of this work boils down to the employment of symbolic methods for the effective and efficient computation of the set containing the boundary unsafe states for any instantiation of the RAS class of Definition II.1, and also the subset of its minimal elements.*

Closing this introductory discussion on the RAS structure and the corresponding problem of deadlock avoidance that are considered in this work, we want to notice that the RAS class introduced in Definition II.1 is known as the class of Disjunctive/Conjunctive (D/C-) RAS in the relevant literature [1]. This class allows for routing flexibility in the sequential logic of the various processes and arbitrary resource allocation requests associated with the various processing stages. On the other hand, this RAS class does not allow for internal cycling in the process routes, merging and/or splitting operations, and any form of uncontrollable behavior. Nevertheless, while we have opted to restrict the subsequent discussion to the class of D/C-RAS for reasons of simplicity and specificity, the algorithms developed herein are straightforwardly extensible to broader RAS classes that exhibit many of the aforementioned behavioral attributes. We shall return briefly to this issue in the closing discussion of Section VI.

III. Modeling the considered RAS as an EFA

A. Extended Finite Automata

An extended finite automaton (EFA) [34] is an augmentation of the ordinary FSA model with integer variables that are employed in a set of guards and are maintained by a set of actions. A transition in an EFA is enabled if and 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.

Definition III.1: An *Extended Finite Automaton (EFA)* over a set of model variables $v = (v_1, \ldots, v_n)$ is a 5-tuple $E = \langle Q, \Sigma, \rightarrow, s_0, Q^m \rangle$ where:

- $Q : L \times \mathcal{D}$ 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* $v = (v_1, \ldots, v_n)$.
- Σ is a nonempty finite set of events (also known as the alphabet of the model).
- $\rightarrow \subseteq L \times \Sigma \times G \times A \times L$ 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.
- $s_0 = (\ell_0, v_0) \in L \times \mathcal{D}$ 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.
- $Q^m \subseteq L^m \times \mathcal{D}^m \subseteq Q$ 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.

In the following, we shall use the notation $\ell \xrightarrow{\sigma}_{g/a} \ell'$ as an abbreviation for $(\ell, \sigma, g, a, \ell') \in \rightarrow$. Also, the symbol ξ will be used to denote neutral actions that do not update the value of the corresponding variables; i.e., if $a_i = \xi$, action a_i does not update the variable v_i in v.

In the EFA modeling framework, the synchronization of two EFAs is formally defined through the operation of the *extended full synchronous composition* (EFSC) of these EFAs.

Definition III.2: Let $E_k = \langle Q_k, \Sigma_k, \rightarrow_k, s_0^k, Q_k^m \rangle$, with $k = 1, 2$, be two EFAs with a common variable set $v = (v_1, \ldots, v_n)$. The *extended full synchronous composition* of E_1 and E_2 is defined as

$$E_1 || E_2 = \langle Q_{1||2}, \Sigma_1 \cup \Sigma_2, \rightarrow_{1||2}, (s_0^1, s_0^2), Q_{1||2}^m \rangle$$

where $Q_{1||2} : L_1 \times L_2 \times \mathcal{D}$, $Q_{1||2}^m : L_1^m \times L_2^m \times \mathcal{D}^m$, $s_0^1 = (\ell_0^1, v_0^1)$ and $s_0^2 = (\ell_0^2, v_0^2)$ with $v_0^1 = v_0^2$, and the conditional transition relation $\rightarrow_{1||2}$ defined as follows:

- For $\sigma \in \Sigma_1 \cap \Sigma_2$, $(\ell_1, \ell_2) \xrightarrow{\sigma}_{g/a} (\acute{\ell}_1, \acute{\ell}_2)$ if $\exists\ \ell_1 \xrightarrow{\sigma}_{g^1/a^1} \acute{\ell}_1 \in \rightarrow_1$, $\exists\ \ell_2 \xrightarrow{\sigma}_{g^2/a^2} \acute{\ell}_2 \in \rightarrow_2$ s.t.
  - $g = g^1 \wedge g^2$,
  - $\forall v_i \in v$, the action function a_i of a is defined as

$$a_i = \begin{cases} a_i^1 & \text{if } \exists v \models g \text{ s.t. } a_i^1(v) = a_i^2(v) \\ a_i^1 & \text{if } a_i^1 \neq \xi \text{ and } a_i^2 = \xi \\ a_i^2 & \text{if } a_i^1 = \xi \text{ and } a_i^2 \neq \xi \\ \xi & \text{if } a_i^1 = \xi \text{ and } a_i^2 = \xi \end{cases}$$

- For $\sigma \in \Sigma_1 \backslash \Sigma_2$, $\langle \ell_1, \ell_2 \rangle \xrightarrow{\sigma}_{g/a} \langle \acute{\ell}_1, \acute{\ell}_2 \rangle$ if $\ell_1 \xrightarrow{\sigma}_{g/a} \acute{\ell}_1 \in \rightarrow_1$ and $\ell_2 = \acute{\ell}_2$;
- For $\sigma \in \Sigma_2 \backslash \Sigma_1$, $\langle \ell_1, \ell_2 \rangle \xrightarrow{\sigma}_{g/a} \langle \acute{\ell}_1, \acute{\ell}_2 \rangle$ if $\ell_2 \xrightarrow{\sigma}_{g/a} \acute{\ell}_2 \in \rightarrow_2$ and $\ell_1 = \acute{\ell}_1$.

Note that when two action functions a_i^1 and a_i^2 update v_i to different values, they are considered as conflicting and we assume that no transition will occur. Furthermore, for the entire EFSC operation to be feasible, EFAs E_1 and E_2 must agree on the pricing of their common variables in their initial states s_0^1 and s_0^2. The EFSC operator is both commutative and associative, and, thus, it can be extended to handle an arbitrary number of EFAs.

B. EFA-based modeling of the considered RAS

This section provides a straightforward procedure for the development of the EFA modeling the behavior of the RAS encompassed in Definition II.1. For the sake of simplicity and clarity, in the following we motivate and illustrate this procedure by developing the EFA model for a simple RAS instance that will be used as an expository example for all the key results of this manuscript. A more formal description of the procedure can be found in the electronic supplement to this paper that is provided at [37]. Also, in the last part

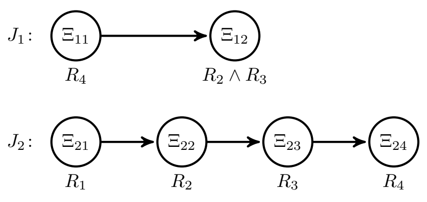

Fig. 1: The RAS configuration considered in Example III.1.

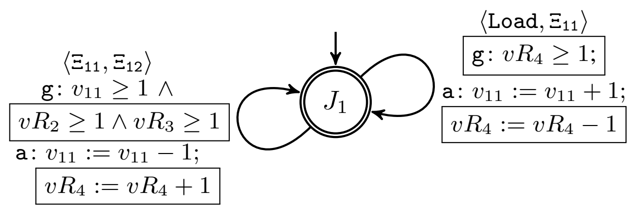
$\langle \Xi_{11}, \Xi_{12} \rangle$
g: $v_{11} \geq 1 \wedge$
$\boxed{vR_2 \geq 1 \wedge vR_3 \geq 1}$
a: $v_{11} := v_{11} - 1;$
$\boxed{vR_4 := vR_4 + 1}$
J_1
$\langle \texttt{Load}, \Xi_{11} \rangle$
g: $\boxed{vR_4 \geq 1;}$
a: $v_{11} := v_{11} + 1;$
$\boxed{vR_4 := vR_4 - 1}$

Fig. 2: The EFA modeling process type J_1.

of the section, some additional remarks elaborate on the informational content of the generated EFA, and on its (proper) interpretation in the context of the algorithmic procedures that are the main theme of this work.

Example III.1: The RAS considered in this example is shown in Fig.1, and it comprises two process types J_1 and J_2. Each process type is defined as a sequence of processing stages; the stages of process type J_1 are denoted by Ξ_{11} and Ξ_{12}, while the stages of process type J_2 are denoted by $\Xi_{21}, \Xi_{22}, \Xi_{23}$ and Ξ_{24}. The set of the system resource types is $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$, with capacities $C_i = 1$ for $i = 1, 2, 3, 4$. In the depicted RAS, stage Ξ_{12} of process type J_1 requests one unit from each of the resource types R_2 and R_3 to properly support its execution. On the other hand, stage Ξ_{11} of process type J_1, and all stages of process type J_2, request only one unit from a single resource type; the corresponding resource types are depicted in Fig.1 next to each of these stages.

Next, we focus on the development of an EFA modeling the behavior of process type J_1. An EFA model for process type J_2 can be developed and interpreted similarly.

Declaration of the resource variables. For each resource type $R_i \in \mathcal{R},\ i = 1, 2, 3, 4$, of the considered RAS, we introduce a *resource variable* vR_i to trace the number of available (or free) units of R_i. The domain of vR_i is $\{0, \ldots, C_i\}$, where C_i is the capacity of R_i and, for this example, it is equal to one. Furthermore, since, under proper RAS operation, the initial and the target states correspond to the RAS empty state, we set the initial and the marked value of each variable vR_i equal to C_i.

Representation of the process sequential logic by a single-location EFA. Next, we proceed to build an EFA that captures the evolution of any process instance from process type J_1 through its various processing stages. The EFA model constructed at this phase concerns only the representation of the routing possibilities of these process instances, and it does not address the relevant resource 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 two transitions correspond to the process-initiation (or loading – $\langle \texttt{load}, \Xi_{11} \rangle$) and the process-advancing ($\langle \Xi_{11}, \Xi_{12} \rangle$) events that appear in the sequential logic of process type J_1. On the other hand, since a process instance that has reached its final stage can always leave the system without posing any further resource requests, the process-termination (or unloading) event is modeled only implicitly through the event $\langle \Xi_{11}, \Xi_{12} \rangle$ that models the process access to its terminal stage. Furthermore, we define a set of *instance variables*, v_{ij}, that count the number of process instances executing at the corresponding processing stages Ξ_{ij} that are explicitly recognized by the model; hence, for this example, only one instance variable, v_{11}, is defined. By making use of this instance variable, we can construct the necessary guards and actions for the EFA transitions. As depicted in Fig. 2, the guards determine whether a process-advancing event can take place, on the basis of the process availability at the originating stage. Upon the occurrence of such an event, the corresponding actions update accordingly the number of the process instances at the involved stages.[4]

Representation of the resource allocation function and its induced dynamics. Fig. 2 also shows the role of the resource allocation variables in the EFA that models the complete behavior of process type J_1. More specifically, the resource allocation requests that are posed by the various processing stages, are modeled by additional guards and actions associated with the corresponding EFA transitions; these guards and actions are highlighted by a boxing frame in Fig. 2. As a more concrete example, consider the transition labeled by $\langle \Xi_{11}, \Xi_{12} \rangle$ in the EFA depicted in Fig. 2. To execute this transition, the associated guard requires not only the presence of an available process instance at stage Ξ_{11}, but also the availability of a free unit from each of the resource types R_2 and R_3. Similarly, upon the execution of this transition, besides the updating of the number of process instances at stage Ξ_{11}, the augmented version of the relevant action function updates also the resource variables to reflect properly the new resource allocation state. For this example, since stage Ξ_{12} is the terminal stage of process type J_1, and the unloading event is only implicitly modeled through event $\langle \Xi_{11}, \Xi_{12} \rangle$, the relevant action function simply releases the previously allocated unit of resource R_1.

The general modeling procedure. Generalizing from the previous example, the basic procedure that converts a given

[4] As already discussed, the considered EFA model for process type J_1 does not avail of a variable v_{12} since it is assumed that a process instance reaching stage Ξ_{12} is (eventually) unloaded from the system, without the need for any further resource allocation action. However, we should further clarify that the omission of the terminal stage Ξ_{12} from the developed EFA model is justified on the assumption that these EFA models of the RAS process types, and the corresponding analysis that is pursued in this paper, focus only on the issue of deadlock avoidance. Terminal processing stages cannot be involved in the formation of deadlock, and therefore, they do not necessitate an explicit consideration. It is implicitly assumed, though, that the system (controller) keeps track of the physical presence of any active process instances in these terminal stages and of any temporary blocking effects that are incurred by this presence. From a more methodological standpoint, the omission of the terminal processing stages from the developed EFA models is in line with the *"projection"* operation that eliminates a subset of processing stages in the DAP synthesis methods that are presented in [23], [41].

RAS instance $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ to the corresponding EFA $E(\Phi)$ modeling the dynamics of Φ, can be summarized by the following three stages: (i) The procedure starts by defining the set of resource variables $\{vR_1, \ldots, vR_m\}$ that monitor the numbers of available units of the resource types $\mathcal{R} = \{R_1, \ldots, R_m\}$ during the evolution of the resource allocation state of Φ. (ii) Subsequently, the sequential logic of each process type J_j in $\mathcal{P}$ is modeled by a single-location EFA E_j. To capture the execution of a single process instance of J_j, a set of variables, v_{jk}, is defined and utilized to construct the necessary guards and actions. These variables are in one-to-one correspondence with the non-terminal stages Ξ_{jk} in the corresponding set $\mathcal{S}_j$, and each of them traces the number of process instances that are executing the corresponding processing stage. The domain of integer values for each instance variable v_{jk} is defined as $\{0, \ldots, \theta_{jk}\}$, and the minimum value of 0 is, both, the initial and the marked value for each v_{jk}. On the other hand, the maximum value θ_{jk} that is associated with instance variable v_{jk} can be set to any upper bound for the number of process instances that can simultaneously execute processing stage Ξ_{jk}. Such a bound, that is implied by the capacities of the resource types utilized by processing stage Ξ_{jk}, can be easily computed as follows:

$$\theta_{jk} = \min_i \{ \left\lfloor \frac{C_i}{\mathcal{A}_{jk}[i]} \right\rfloor : \ \mathcal{A}_{jk}[i] > 0 \}. \tag{1}$$

(iii) In order to represent the dynamics of the resource allocation that takes place at the different processing stages of J_j, the EFA E_j are augmented with the resource variables $\{vR_1, \ldots, vR_m\}$, and the guards and actions of the transitions of E_j are extended to consider and maintain the information that is contained in these new variables. Regarding the maintenance of the resource variables, the augmented EFA implement the following logic: If the executed transition in E_j corresponds to the process-initiating event $\langle \mathtt{J_j\_loading}, \mathtt{\Xi_{jk}} \rangle$, then it is adequate to merely allocate the resources that are necessary for the execution of the initial processing stage Ξ_{jk}. On the other hand, if the transition-labeling event is an event $\langle \mathtt{\Xi_{jk}}, \mathtt{\Xi_{jk'}} \rangle$ advancing an already initiated process instance from its current stage Ξ_{jk} to a subsequent stage $\Xi_{jk'}$, the detailed updating of the resource variables depends on whether stage $\Xi_{jk'}$ is a terminal stage for process type J_j. If it is, there is no need for (explicitly) allocating the resources requested by the executing instances at stage $\Xi_{jk'}$, and the corresponding updating of the resource variables only releases the resources allocated to the advancing process instance while at stage Ξ_{jk}. On the other hand, if stage $\Xi_{jk'}$ is non-terminal, we also need to explicitly allocate the necessary resource units for the execution of this stage, updating accordingly the corresponding resource variables. (iv) Finally, the resource allocation dynamics generated by RAS Φ are formally expressed by the extended full synchronous composition (EFSC) that composes the aforementioned EFA E_j to the "plant" EFA $E(\Phi)$; i.e., $E(\Phi) = E_1 || \ldots || E_n$. The reader is referred to [37] for a more formal description of this entire procedure.

Some further remarks. Closing this section on the representation of the considered RAS dynamics in the EFA modeling framework, we need to elaborate further on the way that these dynamics are captured by the generated EFA $E(\Phi)$, since these elaborations provide necessary context for the algorithms that are developed in the following section.

We begin this discussion by noticing that every legitimate resource allocation state of the considered RAS must adhere to the restrictions that are imposed by the limited capacities of the system resources. In the representation of the EFA $E(\Phi)$, these restrictions are expressed by the following constraints on the pricing of the model variables $v_{jk}, \ j = 1, \ldots, n, \ k = 1, \ldots, l(j)$, and $vR_i, \ i = 1, \ldots, m$:

$$\forall i \in \{1, \ldots, m\}, vR_i + \sum_{j=1}^{n} \sum_{k \in \{1, \ldots, l(j)\} \setminus \mathcal{T}(j)} \mathcal{A}_{jk}[i] * v_{jk} = C_i. \tag{2}$$

In (2), we have taken into consideration the fact that terminal processing stages are not explicitly accounted for in the considered EFA model (for the reasons explained earlier). From a more technical standpoint, the constraints of (2) can be perceived as a set of (resource-induced) *invariants* that must be observed by the dynamics of the EFA $E(\Phi)$ in order to provide a faithful representation of the actual RAS dynamics. Hence, in the following, we shall characterize a state s of the EFA $E(\Phi)$ with a variable vector v satisfying the constraints of (2), as a *feasible* state. Furthermore, the reader can easily verify that any execution of the EFA $E(\Phi)$ that starts from some feasible state s and evolves the state according to the transition-firing logic that is encoded in this automaton, maintains the state feasibility.[5] In particular, since the initial state s_0 of the considered EFA has $v_{jk} = 0, \ \forall j, k$, and $vR_i = C_i, \ \forall i$, s_0 is a feasible state, and all the states that are reachable from it (known as the *reachable state space* of $E(\Phi)$) are also feasible. This last remark implies that the EFA $E(\Phi)$ provides, indeed, a faithful representation of the dynamics of the underlying RAS Φ.

On the other hand, the specification of the state set Q as $Q = L \times \prod_i \mathcal{D}(vR_i) \times \prod_{j,k} \mathcal{D}(v_{jk})$, where the domain sets $\mathcal{D}(v)$ of the various variables v are determined as described in the previous paragraphs, implies that Q may contain infeasible states, as well. Since these states remain unreachable in any proper execution of $E(\Phi)$, they do not compromise the analytical power of this model regarding the traced RAS dynamics. Yet, these states might still be a nuisance in the computations that are effected by the presented algorithms, since they might encumber these computations with excursions to state regions that are irrelevant to the actual system dynamics. Whenever this problem arises, it can be addressed by "filtering" the relevant state subsets for state feasibility. In Section IV we shall provide a more specific implementation of such a filtering mechanism that is appropriate for the symbolic computation that is pursued in that section.

Example III.2: Fig. 3 depicts the dynamic behavior encoded by the EFA $E(\Phi)$ that was developed for the RAS instance in Example III.1. The depicted state transition diagram (STD) includes only the RAS feasible states that are reachable from the initial state of $E(\Phi)$, and furthermore, it considers only those states that are modeled explicitly in this EFA through

[5]A formal proof of this fact can be easily constructed as an inductive argument that is based on the length of the executed trace.

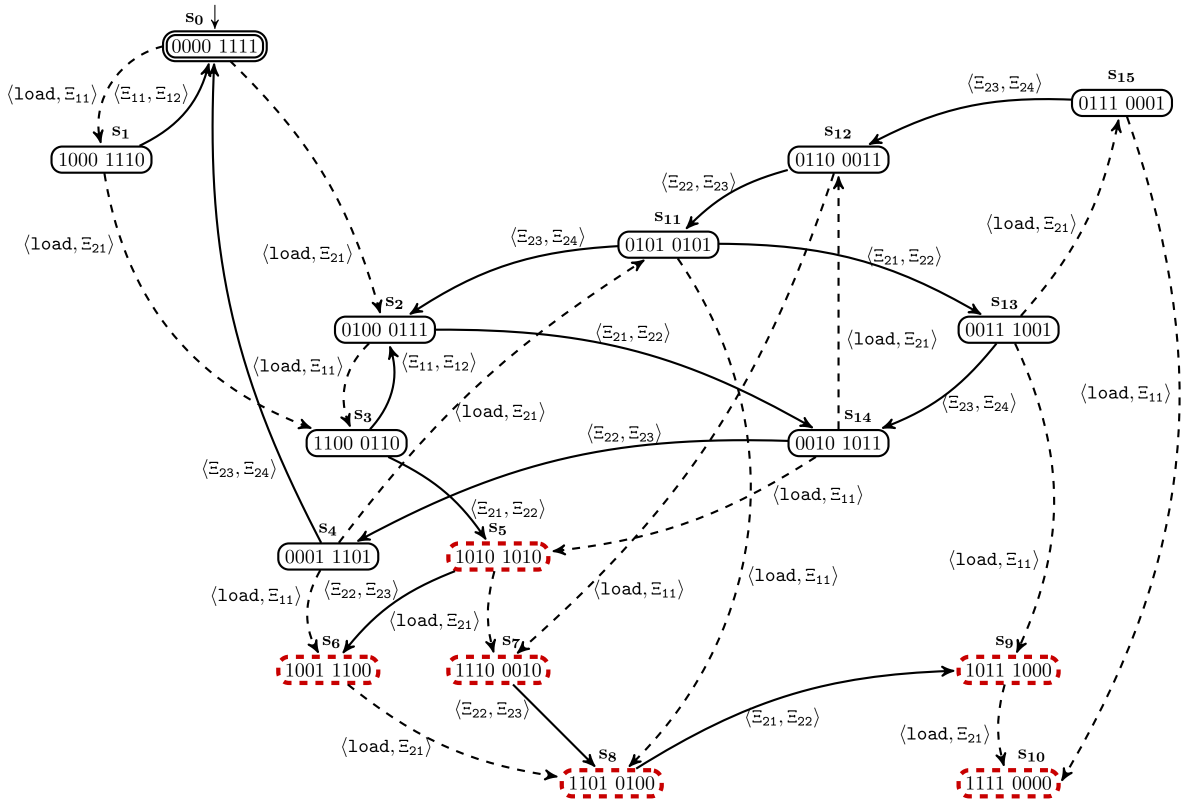

Fig. 3: The state transition diagram (STD) modeling the RAS dynamics encoded by the EFA $E(\Phi)$ that was developed for Example III.1.

the pricing of the corresponding model variables. As it can be seen in Fig. 3, the considered STD involves sixteen (16) states, denoted by $\mathbf{s}_i$, where $i = 0, \ldots, 15$. Each state is described by eight components that correspond to the values of the instances variables $v_{11}, v_{21}, v_{22}, v_{23}$ and the resource variables vR_1, vR_2, vR_3, vR_4 of the EFA $E(\Phi)$. States depicted by a dashed line are unsafe. Furthermore, for reasons that will become clear in the following, the transitions in the depicted STD are partitioned into two subsets that collect respectively the transitions corresponding to "process initiation" (or "loading") and "process advancement" events. These two subsets of transitions are depicted respectively as dashed and solid transitions in the STD of Fig. 3.

IV. Computing the minimal boundary unsafe states

A. Binary Decision Diagrams and Symbolic Computation

Binary decision diagrams (BDDs) [27] are a memory-efficient data structure used to represent Boolean functions as well as to perform set-based operations. The field in computer science that studies the employment of BDDs in the support of the aforementioned tasks has come to be known as "symbolic computation", and a systematic exposition of the relevant theory can be found, for instance, in [42], [43]. In the next paragraphs we introduce some theoretical concepts and elements pertaining to BDDs and symbolic computation that provide necessary background and context for the main developments of this section.

To present the basic BDD theory employed in this work, in the following, we set $\mathbb{B} \equiv \{0, 1\}$. Also, for any Boolean function $f\colon \mathbb{B}^n \to \mathbb{B}$, in n Boolean variables $X = (x_1, \ldots, x_n)$, we denote by $f|_{x_i=0(\text{resp. }1)}$ the Boolean function that is induced from function f by fixing the value of variable x_i to 0 (resp. 1). Then, a BDD-based representation of f is a graphical representation of this function that is based on the following identity:

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

More specifically, (3) enables the representation of the Boolean function 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 for any given pricing of the variable set X 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 the aforementioned 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 power of BDDs lies in the efficiency that they provide in the execution of binary operations. Let f and f' be two Boolean functions of X. Then, it should be evident from (3) that a binary operator $\otimes$ between (the BDDs representing) f and f' can be recursively computed as

$$f \otimes f' = [\neg x \wedge (f|_{x=0} \otimes f'|_{x=0})] \vee [x \wedge (f|_{x=1} \otimes f'|_{x=1})] \quad (4)$$

where $x \in X$. If dynamic programming is used, the computation implied by (4) can have a complexity of $\mathcal{O}(|f| \cdot |f'|)$ where $|f|$ and $|f'|$ are the sizes of (the BDDs representing) f and f'.

A particular operator that is used extensively in the following is the *existential quantification* of a function f over its Boolean variables. For a variable $x \in X$, the existential quantification of f is defined by $\exists x.f = f|_{x=0} \vee f|_{x=1}$. Also, if $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_k) \subseteq X$, then $\exists \bar{X}.f$ is a shorthand notation for $\exists \bar{x}_1.\exists \bar{x}_2. \ldots \exists \bar{x}_k.f$. In plain terms, $\exists \bar{X}.f$ denotes all those truth assignments of the variable set $X \backslash \bar{X}$ that can be extended over the set $\bar{X}$ in a way that function f is eventually satisfied.

B. EFA encoding through BDDs.

To represent an EFA E by a Boolean function, different sets of Boolean variables are employed to encode the locations, events and integer variables. For the encoding of the state set $Q : L \times \mathcal{D}$, 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 $q \notin \bar{Q}$.[6] 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

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

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

Given a RAS instance Φ and the distinct EFA $E_1, \ldots, E_n$ that model the resource allocation dynamics of the RAS process types $J_1, \ldots, J_n$, we shall denote by $\Delta_1, \ldots, \Delta_n$ the corresponding symbolic representations of those EFA. Furthermore, we shall denote by $\Delta_{\mathbf{E}}$ the symbolic representation of $E(\Phi)$, the EFA that models the integrated dynamics of RAS Φ. We remind the reader that $E(\Phi)$ is defined as the EFSC of the EFA $E_1, \ldots, E_n$. Hence, Δ_E can be systematically obtained from $\Delta_1, \ldots, \Delta_n$ through the approach for the symbolic computation of EFSC that is presented in [33]; we refer the reader to that work for the relevant details.

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

We also remind the reader that, while Δ_E provides a symbolic representation of the resource allocation dynamics of RAS Φ, it also contains a subset of infeasible states that were described in the closing remarks of Section III. The infeasibility of these states can be detected by the characteristic function χ_F that expresses state feasibility in the BDD-based representational context, and it can be constructed as follows: First, the invariants of (2) are collectively expressed by the following Boolean function

$$\bigwedge_{i=1}^{m} \Big(vR_i + \sum_{j=1}^{n} \sum_{k \in \{1,\ldots,l(j)\} \backslash \mathcal{T}(j)} \mathcal{A}_{jk}[i] * v_{jk} = C_i\Big). \quad (5)$$

Then, χ_F is defined by the BDD that collects the binary representations of all the value sets for the variables vR_i and v_{jk} that satisfy the Boolean function of (5). As a more concrete example, the instantiation of (5) for the example RAS of Fig. 1 is as follows:

$$\begin{aligned} &1 * v_{11} + vR_4 = 1 \ \wedge \ 1 * v_{21} + vR_1 = 1 \ \wedge \\ &1 * v_{22} + vR_2 = 1 \ \wedge \ 1 * v_{23} + vR_3 = 1. \end{aligned} \quad (6)$$

Finally, as it will be revealed in the following, the computations pursued in this work do not require the explicit representation of the event set $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$. Hence, to reduce the number of Boolean variables employed by $\Delta_{\mathbf{E}}$, in the following we will suppress from $\Delta_{\mathbf{E}}$ the Boolean variable set $X^{\Sigma}_{\mathbf{E}}$, that represents Σ. In addition, since the locations of the considered EFA do not convey any substantial information other than characterizing the various process types as model entities with a distinct behavior modeled by the corresponding EFA, $\Delta_{\mathbf{E}}$ can be further compressed by suppressing the Boolean variable set $X^L_{\mathbf{E}} = X^L_1 \cup \ldots \cup X^L_n$, as well. The elimination of the aforementioned sets of variables from $\Delta_{\mathbf{E}}$ is technically effected through the following existential quantification:

$$\Delta_{\mathbf{E}} := \exists (X^{\Sigma}_{\mathbf{E}} \cup X^L_{\mathbf{E}}).\Delta_{\mathbf{E}} \quad (7)$$

In the rest of this work, when we refer to the plant model $\Delta_{\mathbf{E}}$ we shall imply the output of the operation performed in (7).

The following three subsections assume the availability of an appropriately constructed BDD $\Delta_{\mathbf{E}}$ that is a valid representation of the composed EFA $E(\Phi) = E_1 || \ldots || E_n$ and has been compressed through the existential quantification expressed in (7), and proceed to present a series of symbolic algorithms for the computation of the set of (minimal) boundary unsafe states in the considered RAS. As remarked in Section I, the presented developments provide two different algorithms for retrieving all the boundary unsafe states within the underlying RAS state-space, and an additional algorithm for identifying and removing non-minimal elements from this set.

Algorithm 1: Symbolic computation of the reachable boundary unsafe states

```
Input: $\Delta_{\mathbf{E}}$ and $\chi_{\{s_0\}}$
Output: $\chi_{RB}$
1  $\chi_R := \chi_{\{s_0\}}$
2  repeat            // compute the set of reachable states, $\chi_R$
3      $\chi_{R_{pre}} := \chi_R$
4      $\chi_{R_{cur}} := \exists X^{\mathcal{D}}.(\chi_R \wedge \Delta_{\mathbf{E}})$
5      $\chi_R := \chi_{R_{pre}} \vee (\chi_{R_{cur}}[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}])$
6  until $\chi_R = \chi_{R_{pre}}$
7  $\chi_C := \chi_{\{s_0\}}$
8  repeat            // compute the set of safe states, $\chi_C$
9      $\chi_{C_{pre}} := \chi_C$
10     $\chi_{C_{cur}} := \exists \acute{X}^{\mathcal{D}}.(\chi_C[X^{\mathcal{D}} \to \acute{X}^{\mathcal{D}}] \wedge \Delta_{\mathbf{E}})$
11     $\chi_C := \chi_{C_{pre}} \vee \chi_{C_{cur}}$
12 until $\chi_C = \chi_{C_{pre}}$
13 $\chi_{RC} := \chi_R \wedge \chi_C$// compute the reachable $\wedge$ safe state set
   // finally, compute the set of boundary unsafe states, $\chi_{RB}$
14 $\Delta_{\hat{B}} := \chi_{RC} \wedge \Delta_{\mathbf{E}}$
15 $\chi_{\hat{B}} := (\exists X^{\mathcal{D}}.\Delta_{\hat{B}})[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]$
16 $\chi_{RB} := \chi_{\hat{B}} \wedge \neg\chi_C$
```

C. An extension of the standard SCT synthesis algorithm for the computation of reachable boundary unsafe states

The first algorithm for the computation of the RAS boundary unsafe states that is developed in this work is depicted in Algorithm 1, and it constitutes an adaptation of the general algorithm that has been developed by the SCT for supporting maximally permissive non-blocking supervision. More specifically, given the symbolic representation of the composed transition relation $\Delta_{\mathbf{E}}$ that is defined by (7), and the corresponding characteristic function $\chi_{\{s_0\}}$ representing the initial and marked state of the EFA $E(\Phi)$,[7] Algorithm 1 computes the characteristic function of the reachable boundary unsafe state set, χ_{RB}, through the following symbolic operations.

The algorithm starts with the computation of a symbolic representation of the reachable state set χ_R, through the forward search depicted in Lines 1-6. For that, the algorithm employs two characteristic functions $\chi_{R_{pre}}$ and $\chi_{R_{cur}}$ to respectively represent (i) the state set χ_R already reached at the beginning of each iteration of the forward-search process, and (ii) the set of states that can be reached from the current elements of χ_R through a single transition of $\Delta_{\mathbf{E}}$. The reachable state set χ_R keeps expanding with the new states entering $\chi_{R_{cur}}$ at each iteration, until no new reachable state can be computed. At Line 5, the operation $[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]$ denotes the replacement of all variables of $\acute{X}^{\mathcal{D}}$ by those of $X^{\mathcal{D}}$, so that the reachable states identified at each iteration are eventually represented by $X^{\mathcal{D}}$ and the forward search can continue. The characteristic function of the co-reachable state set, denoted by χ_C, can be computed in a similar manner. The corresponding backward search is depicted in Lines 7-12 of Algorithm 1.

As remarked in Section II, in the RAS literature, co-reachable states are also referred to as safe states, and states that are not co-reachable are characterized accordingly as unsafe. The set B of *boundary* unsafe states can be formally expressed as $B \equiv \{u \mid \exists\ s \to u$ in $\Delta_{\mathbf{E}}$ s.t. $s \in \chi_C$ and $u \notin \chi_C\}$. Having obtained the characteristic functions χ_R and χ_C, the characteristic function of the reachable safe state set, χ_{RC}, can be obtained through the conjunction depicted at Line 13. On the other hand, to compute the characteristic function of the reachable boundary unsafe state set χ_{RB}, Algorithm 1 first retrieves from $\Delta_{\mathbf{E}}$ all the transitions with their source state belonging to χ_{RC}. The set of these retrieved transitions is denoted by $\Delta_{\hat{B}}$, and its computation is carried out in Line 14 of the algorithm. Subsequently, Line 15 collects in $\chi_{\hat{B}}$ the set of the target states of the transitions extracted in $\Delta_{\hat{B}}$. Finally, in Line 16, the characteristic function χ_{RB} is computed by removing from set $\chi_{\hat{B}}$ all the safe states, i.e., all those states that also belong in χ_C.

It is clear from the above discussion that Algorithm 1 terminates in finite time. Also, since this algorithm relies extensively on standard procedures developed by SCT for establishing maximally permissive non-blocking supervision, a formal proof for its correctness can be based on arguments provided in the corresponding SCT literature, and we refer to that literature for the relevant details (c.f., for instance, [12], [13], [33]).

Example IV.1: As a concrete example, we apply Algorithm 1 to the STD depicted in Fig. 3. The transitions of the STD are symbolically represented in the BDD $\Delta_{E(J_1)||E(J_2)}$, where $E(J_1)$ and $E(J_2)$ are the EFA modeling the process types J_1 and J_2 of the RAS instance depicted in Fig. 1. Starting with $\chi_R := \{\mathrm{s}_0\}$, the computation of Lines 2-6 will return the set χ_R containing all of the sixteen reachable states s_0 – s_{15}. On the other hand, the set χ_C obtained from the computation in Lines 7-12 will contain all the safe states s_0 – s_4, s_{11} – s_{15} depicted in Fig. 3, and possibly some additional safe but unreachable states (not depicted in Fig. 3). The subsequent conjunction of χ_C with χ_R in Line 13 filters out the unreachable safe states from χ_C; i.e., the returned set χ_{RC} contains only the reachable and safe states s_0 – s_4, s_{11} – s_{15} depicted in Fig. 3.[8] Finally, the algorithm operations in Lines 14-16 will return the set χ_{RB} containing the states s_5 – s_{10}, i.e., all the reachable and unsafe states depicted in Fig. 3, since all these states can be reached from some reachable safe state in χ_{RC} in one transition.

[7]Since the Boolean variable set representing the locations of the original EFA $\Delta_{\mathbf{E}}$ has been existentially quantified, s_0 is just equal to the vector of the initial (and also the marked) values for the variables.

[8]In fact, the EFA $E(\Phi)$ corresponding to the RAS considered in this example does not contain any feasible unreachable states. This can be established by noticing that in the considered RAS class, any feasible unreachable states essentially result by "swapping" (i.e., simultaneously advancing) process instances that are in deadlock. But in the semantics of the EFA $E(\Phi)$ of Section III, any deadlock must involve some process instance of type J_1, and the aforementioned swapping of these process instances implies their immediate unloading from the system.

D. An alternative algorithm for the computation of feasible boundary unsafe states

In this subsection, we present an alternative symbolic algorithm that decomposes the computation of the boundary unsafe states into two stages. In the first stage, all the deadlock states w.r.t. the advancement events in the considered RAS are identified and computed from the symbolic representation of the state space, $\Delta_{\mathbf{E}}$. 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. The entire computation is formally expressed by Algorithm 2, that works with the BDDs of $\Delta_{\mathbf{E}}$ and the characteristic function χ_F,[9] and returns the characteristic functions χ_{FD} and χ_{FB} that constitute respective symbolic representations of the sets of the feasible deadlock states and the feasible boundary unsafe states. In general, the set χ_{FB} obtained from the presented algorithm may include some states that are not reachable from the initial state s_0; i.e., in general, $\chi_{RB} \neq \chi_{FB}$ but $\chi_{RB} \wedge \chi_{FB} = \chi_{RB}$. Hence, the presence of any additional states in the set χ_{FB} does not impede the implementation of the maximally permissive DAP by means of this set and the one-step-lookahead logic that was outlined in the earlier parts of this manuscript. Furthermore, for reasons that will become clear in the following, it is pertinent to assume that the characteristic function $\Delta_{\mathbf{E}}$ is partitioned in the characteristic functions Δ_A and Δ_L that collect respectively the transitions in $\Delta_{\mathbf{E}}$ corresponding to process advancement and process loading events; obviously, $\Delta_{\mathbf{E}} = \Delta_A \vee \Delta_L$. The rest of this section elaborates on the various phases of the computation that is depicted in Algorithm 2, and establishes formally its correctness.

Identification of the feasible deadlock states. The symbolic operations for the computation of the characteristic function χ_{FD} are depicted in Lines 1-4 of Algorithm 2, and they can be described by the following two steps:
1) The first step consists of Lines 1-3 in Algorithm 2 and it computes the characteristic function χ_D of all the (partial) deadlock states in $\Delta_{\mathbf{E}}$, i.e., those states that are different from the initial state s_0 and they do not enable any process-advancing events. This function is computed by first extracting into the characteristic function χ_T all the target states from $\Delta_A \vee \Delta_L$ (i.e. from $\Delta_{\mathbf{E}}$) and in the characteristic function χ_E all the states that enable process-advancing events. Subsequently, χ_D is computed as the elements of χ_T that are not in χ_E (i.e., they do not enable any process-advancing events) or the initial state s_0.
2) Since χ_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 resource-induced invariants of (2)). The presence of these infeasible states in χ_D would increase unnecessarily the computational cost of the second stage of the considered algorithm, that utilizes the identified deadlock states as starting points for the identification of the additional set of deadlock-free unsafe states. Hence, in the last step of the first stage of Algorithm 2, the obtained state set χ_D is filtered through its conjunction with the characteristic function χ_F in order to obtain the set of feasible deadlock states; this set is represented by the characteristic function χ_{FD}.

Example IV.2: The application of Lines 1-4 of Algorithm 2 to the BDDs Δ_A and Δ_L corresponding to the STD depicted in Fig. 3, will return the BDD of feasible deadlock states, χ_{FD}, that includes the states s_6, s_9 and s_{10}. Indeed, it can be clearly seen in the depicted STD that no solid edges emanate from these three states; i.e., these states enable no process-advancing events. The reader should also notice that states $\mathbf{s_6}$ and $\mathbf{s_9}$ *do* enable process-loading events; but these events do not contribute to the progress of the already initiated process instances, and they only aggravate an already problematic situation. Finally, we also remind the reader that, according to Footnote 8, the EFA $E(\Phi)$ corresponding to this example does not contain any feasible unreachable states, and therefore, the aforementioned three states is the entire content of the state-set encoded by the BDD χ_{FD} that is returned at Line 4.

Computation of the feasible boundary unsafe states. Having obtained the set χ_{FD} of the feasible deadlock states, the algorithm proceeds with the symbolic computation of the feasible boundary unsafe states in the RAS state-space $\Delta_{\mathbf{E}}$. These states are collected in the characteristic function χ_{FB}, which is computed in Lines 5-18 of Algorithm 2. A detailed description of this computation is as follows:
1) At this phase of the computation, Algorithm 2 employs the set U in order to collect all the identified unsafe states. Furthermore, 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 Δ_A, in an effort to reach and explore new states. The corresponding symbolic representations for these two sets, denoted by χ_U and $\chi_{U_{new}}$, are initialized to χ_{FD}. Finally, we also define the transition set $\hat{U}_{pre} \equiv \{(s,u) \in \Delta_A \mid u \in U \wedge s \notin U\}$; i.e., 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. The characteristic function of $\hat{U}_{pre}$ is initialized to zero.
2) During the main iteration of the executed search process, the algorithm first extracts all the states that can be reached from the unsafe state set U_{new} by tracing backwards some process-advancing transition in Δ_A. This computation is performed in Lines 7-8 of the algorithm, with the extracted states represented by the characteristic function $\chi_{S\hat{U}}$. Also, the backtraced transitions of Δ_A are represented by the characteristic function $\Delta_{\hat{U}}$.
3) Subsequently, Algorithm 2 tries to resolve which of the states collected in $\chi_{S\hat{U}}$ can be classified as unsafe. This resolution is performed in Lines 9-11 of the algorithm. More specifically, the algorithm first collects in the transition set Δ_{SA} all those process-advancing transitions of Δ_A that emanate from states in $\chi_{S\hat{U}}$. Subsequently, it removes from Δ_{SA} those transitions that are known to lead to unsafe states, namely the transitions that are also in $\Delta_{\hat{U}}$ and in $\hat{U}_{pre}$. The source states for any transitions remaining in Δ_{SA} after this last operation are collected in χ_{NU}; these are states that have transitions leading to states currently not in U, and therefore, they cannot be classified as unsafe (at least in this iteration).

[9]We remind the reader that χ_F characterizes the feasibility of the various states encountered in $\Delta_{\mathbf{E}}$ w.r.t. the resource-induced invariants of (2).

Algorithm 2: Symbolic computation of the boundary unsafe states

```
Input: $\Delta_{\mathbf{E}}$ (as $\Delta_A \vee \Delta_L$) and $\chi_F$
Output: $\chi_{FB}$
/* Compute the feasible deadlock states $\chi_{FD}$ */
1  $\chi_T := \big(\exists X^{\mathcal{D}}.\ (\Delta_A \vee \Delta_L)\big)[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]$   // $\chi_T$ collects the target states in the transitions of $\Delta_{\mathbf{E}}$;
2  $\chi_E := \exists \acute{X}^{\mathcal{D}}.\ \Delta_A$   // $\chi_E$ contains the states of $\Delta_A$ that enable advancement events;
3  $\chi_D := \chi_T \wedge \neg\chi_E \wedge \neg\chi_{\{s_0\}}$   // $\chi_D$ is the set of deadlock states w.r.t. advancement events, including infeasible states;
4  $\chi_{FD} := \chi_D \wedge \chi_F$   // $\chi_{FD}$ is the set of feasible deadlock states w.r.t. advancement events;
/* Compute the feasible boundary unsafe states $\chi_{FB}$ from $\chi_{FD}$ */
5  $\chi_{U_{new}} := \chi_{FD}, \chi_U := \chi_{FD}, \Delta_{\hat{U}_{pre}} := 0$   // initialization;
6  repeat
7      $\Delta_{\hat{U}} := \chi_{U_{new}}[X^{\mathcal{D}} \to \acute{X}^{\mathcal{D}}] \wedge \Delta_A$   // $\Delta_{\hat{U}}$ contains the transitions in $\Delta_A$ where the target states belong to $\chi_{U_{new}}$;
8      $\chi_{S\hat{U}} := \exists \acute{X}^{\mathcal{D}}.\ \Delta_{\hat{U}}$   // $\chi_{S\hat{U}}$ contains the source states of $\Delta_{\hat{U}}$;
9      $\Delta_{SA} := \chi_{S\hat{U}} \wedge \Delta_A$   // $\Delta_{SA}$ contains the transitions in $\Delta_A$ where the source states belong to $\chi_{S\hat{U}}$;
10     $\chi_{NU} := \exists \acute{X}^{\mathcal{D}}.\ (\Delta_{SA} \wedge \neg\Delta_{\hat{U}} \wedge \neg\Delta_{\hat{U}_{pre}})$   // $\chi_{NU}$ contains the states in $\chi_{S\hat{U}}$ that are not qualified as
                                                                     // unsafe states at the current iteration;
11     $\chi_{U_{cur}} := \chi_{S\hat{U}} \wedge \neg\chi_{NU}$   // $\chi_{U_{cur}}$ contains the unsafe states at the current iteration;
12     $\chi_{U_{new}} := \chi_{U_{cur}} \wedge \neg\chi_U$   // $\chi_{U_{new}}$ contains the newly computed unsafe states, which are used for the next iteration;
13     $\chi_U := \chi_U \vee \chi_{U_{cur}}$   // $\chi_U$ accumulates the unsafe states in $\chi_{U_{cur}}$;
14     $\Delta_{\hat{U}_{pre}} := (\Delta_{\hat{U}_{pre}} \vee \Delta_{\hat{U}}) \wedge \neg\chi_{U_{cur}}$   // $\Delta_{\hat{U}_{pre}}$ is updated by first adding the transitions in $\Delta_{\hat{U}}$ and then
                                                                     // removing the transition with the source states in $\chi_{U_{cur}}$;
15 until $\chi_{U_{new}} = 0$
16 $\Delta_{\mathcal{B}} := \chi_U[X^{\mathcal{D}} \to \acute{X}^{\mathcal{D}}] \wedge \Delta_{\mathbf{E}}$   // $\Delta_{\mathcal{B}}$ contains the transitions with the target states belonging to $\chi_U$;
17 $\Delta_{S\mathcal{B}} := \Delta_{\mathcal{B}} \wedge (\neg\chi_U)$   // $\Delta_{S\mathcal{B}}$ contains the transitions in $\Delta_{\mathcal{B}}$ where the source states are safe states;
18 $\chi_{FB} := (\exists X^{\mathcal{D}}.\ \Delta_{S\mathcal{B}})[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]$   // $\chi_{FB}$ is obtained by extracting the set of target states of $\Delta_{S\mathcal{B}}$;
```

On the other hand, the complement of χ_{NU} w.r.t. the overall set of extracted states $\chi_{S\hat{U}}$ must contain states with all their emanating transitions leading to unsafe states, and therefore, they are themselves unsafe; these states are identified and collected in set $\chi_{U_{cur}}$ in Line 11.

4) Lines 12-14 perform the necessary updates so that all the critical data structures represent correctly the current outcome of the ongoing search process. Hence, Line 12 removes from $\chi_{U_{cur}}$ any states that have already been classified as unsafe in the previous iterations; the remaining states are the elements of U_{new} for the next iteration. Line 13 adds to the set U the newly identified unsafe states, and finally, Line 14 updates the transition set $\hat{U}_{pre}$; this last update is performed by initially adding to $\hat{U}_{pre}$ all the transitions in $\Delta_{\hat{U}}$ (i.e., the transitions that were backtraced during the current iteration), and subsequently removing those transitions with source states identified as unsafe.

5) The iteration described in items (2-4) above terminates when no new unsafe states can be identified by the algorithm. At this point, Algorithm 2 proceeds to extract the boundary unsafe states from set χ_U. For that, at Line 16, the algorithm computes from $\Delta_{\mathbf{E}}$ all the transitions with the target states belonging to the unsafe state set χ_U; the relevant transition set is denoted by $\Delta_{\mathcal{B}}$. Next, at Line 17, 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, χ_{FB} is obtained by extracting the target states from $\Delta_{S\mathcal{B}}$ and performing the replacement of $\acute{X}^{D}$ by $X^{\mathcal{D}}$.

Example IV.3: In the context of the example STD of Fig. 3, the backward search of Algorithm 2 in order to identify all the deadlock-free unsafe states, implemented by Lines 6-15, will start from the identified deadlock states $\mathbf{s}_6$, $\mathbf{s}_9$ and $\mathbf{s}_{10}$, and it will be performed on the solid transitions of this STD, i.e., on transitions corresponding to process-advancing events. More specifically, at the first iteration of this search, states $\mathbf{s}_5$ and $\mathbf{s}_8$ are reached by respectively backtracing from states $\mathbf{s}_6$ and $\mathbf{s}_9$, and they are classified as unsafe since the backtraced transitions leading to these states are the only process-advancing transitions emanating from them. The second iteration sets $\chi_{U_{new}}; = \{\mathbf{s}_5, \mathbf{s}_8\}$, and it tries to backtrace through process-advancing transitions from these two states, in quest of new unsafe states. Indeed, this backtracing from state $\mathbf{s}_8$ exposes the unsafety of state $\mathbf{s}_7$. On the other hand, state $\mathbf{s}_3$ that is reached through the backtracing from state $\mathbf{s}_5$ cannot be claimed as unsafe, since $\mathbf{s}_3$ also avails of transition (Ξ_{11}, Ξ_{12}) leading to state $\mathbf{s}_2$, which is not included in the current set of unsafe states. The backward search terminates after the third iteration since the attempt to backtrace from state $\mathbf{s}_7$ – i.e., the contents of the set $\chi_{U_{new}}$ in this iteration – through some process-advancing transitions fails to reach any states at all (and therefore, there are no newly identified states with all of their process-advancing transitions leading to the unsafe states that were identified during the first two iterations). Finally, Algorithm 2 proceeds to extract the boundary unsafe states. Since, in this example, all unsafe states $\mathbf{s}_5 - \mathbf{s}_{10}$ can be reached from the safe subspace through a single transition, the resulting state set χ_{FB} contains all states $\mathbf{s}_5 - \mathbf{s}_{10}$. It is also important to notice that, in the considered example, we were able to identify correctly all the unsafe states without visiting at all the safe states $\mathbf{s}_0 - \mathbf{s}_2$, $\mathbf{s}_4$ and $\mathbf{s}_{11} - \mathbf{s}_{15}$; even for this small example, this unvisited area constitutes more than 50% of the entire state space.

Correctness analysis. To prove the effectiveness of Algorithm 2 w.r.t. the penultimate objective of the implementation of the maximally permissive DAP through the one-step-lookahead scheme that was outlined in the earlier parts of this manuscript, we need to show that (i) the algorithm terminates in a finite number of steps, (ii) the returned set χ_{FB} contains all the feasible boundary unsafe states, and furthermore, (iii) χ_{FB} does not contain any feasible safe state.

The finiteness of Algorithm 2 depends on whether the backward search performed in Lines 6-15 can terminate in a finite number of iterations. We notice that the termination of this search is determined by the set of the new unsafe states, $\chi_{U_{new}}$, computed at each iteration; if $\chi_{U_{new}}$ is empty, the backward search terminates. We also notice that the set $\chi_{U_{new}}$ will finally be empty during the search, since the set of states in Δ_A is finite. Hence, Algorithm 2 terminates in a finite number of steps.

The next theorem establishes the correctness of Algorithm 2, by establishing items (ii) and (iii) in the above list.

Theorem IV.1. *The set χ_{FB} returned by Algorithm 2 possesses the following properties:*

1) *It contains only feasible states.*
2) *It contains all the feasible boundary unsafe states in the underlying RAS state-space.*
3) *It contains no feasible non-boundary unsafe state.*
4) *It contains no safe state.*

Proof: Due to space considerations, here we provide a sketch for the proof of Theorem IV.1. A complete correctness analysis for Algorithm 2, including a more formal proof for the above theorem, can be found in the electronic supplement of [37].

The technical analysis of Algorithm 2 begins by establishing that the characteristic function χ_{FD}, obtained from the symbolic operations performed in Lines 1-4 of Algorithm 2, identifies correctly the feasible deadlock states in the EFA that is represented by the employed BDD $\Delta_{\mathbf{E}}$.

Furthermore, since the transitions that are encoded by the EFA Δ_E observe the invariants of (2), and the set χ_{FD} that is obtained in the first computational phase of Algorithm 2 contains only feasible deadlock states, it follows that all the states that are reached by Algorithm 2 during its second phase of backward search, are also feasible. This last remark establishes the first result of Theorem IV.1.

To show the validity of the remaining clauses of Theorem IV.1, we first establish that the set U maintained by Algorithm 2 will contain, upon the algorithm termination, all the feasible unsafe states and no feasible safe states of $\Delta_{\mathbf{E}}$.

Hence, to show that U will contain all the feasible unsafe states of $\Delta_{\mathbf{E}}$, we first notice that the finite and acyclic nature of the paths that define the execution logic of the various process types in the considered RAS class, imply that the subspace that is reached from any unsafe state u following only transitions in Δ_A has a finite, acyclic structure. This remark, when combined with the presumed unsafety of state u, further implies that every path in Δ_A that emanates from state u is an acyclic path that terminates at some feasible deadlock state. Let ζ denote the longest length of these paths, where the length of a path is defined by the number of the involved transitions. Then, the inclusion in the set U of unsafe states u with $\zeta = 1$ results immediately from the basic operations in the backtracing iteration of Algorithm 2 and the already established fact that the algorithm recognizes successfully all the feasible deadlocks during its first phase. The inclusion in the set U of unsafe states u with $\zeta > 1$ can be addressed by an inductive argument on ζ.

On the other hand, the fact that the state set U does not contain any feasible safe states of $\Delta_{\mathbf{E}}$ can be established by a simple induction on the number of iterations that are performed by the algorithm, while considering the role that is played by the set $\hat{U}_{pre}$ in the algorithm computations.

In view of the above results, to obtain result #2 of Theorem IV.1 it suffices to show that the construction of the set FB in Lines 16-18 of Algorithm 2 retains all the boundary feasible unsafe states in U. But this can be easily checked from the facts that (a) the transition set $\Delta_{\mathcal{B}}$ contains all the transitions with target states in set U, while (b) the transition set $\Delta_{\mathcal{SB}}$ is obtained from $\Delta_{\mathcal{B}}$ by removing only transitions with source states in U (and therefore, unsafe, according to the previous discussion).

Result #3 of Theorem IV.1 is also inferred from the construction of the set $\Delta_{\mathcal{SB}}$ from the set $\Delta_{\mathcal{B}}$ through the removal of all those transitions with source states in U, upon noticing that U contains all the feasible unsafe states of $\Delta_{\mathbf{E}}$.

Finally, Result #4 of Theorem IV.1 is obtained from the facts that the state set U does not contain any feasible safe states and all the transitions in the set $\Delta_{\mathcal{B}}$ have target states in U. ■

E. Computing the minimal boundary unsafe states

An important implication of the invariants of (2) is that, at any feasible state of the RAS state-space, the values of the resource variables vR_i can be induced from the values of the instance variables v_{jk}. In other words, any feasible state s of the considered RAS can be uniquely determined only by the specification of its instance variables v_{jk}. Hence, one can obtain a more compact symbolic representation of the set of feasible boundary unsafe states, χ_{FB}, that is computed by Algorithm 2, by eliminating from the elements of χ_{FB} the values that correspond to the variables vR_i.[10] Letting X^R

[10]We notice, however, that the presence of the variables vR_i during the execution of Algorithm 2 is instrumental for ensuring some of the properties that were established in Theorem IV.1, especially Property 1. Also, while we carry out the subsequent discussion in the context of the set FB that is returned by Algorithm 2, similar remarks and techniques apply to the set RB that is returned by Algorithm 1.

denote the Boolean variables representing the values of the resource variables $vR_i,\ i = 1, \ldots, m$, this elimination can be performed through the following existential quantification:

$$\chi_{FB} := \exists X^R.\ \chi_{FB}. \tag{8}$$

The compressed representation of the set χ_{FB} that is obtained through (8) becomes even more important when noticing that, according to [41], state unsafety is a monotone property in this representation. More specifically, given any two feasible boundary unsafe states u_1, u_2 represented according to the logic of (8), we consider the ordering relation "$\leq$" on them that is defined by the application of this relation componentwise; i.e.,

$$u_1 \leq u_2 \Longleftrightarrow (\forall k = 1, \ldots, K, u_1[k] \leq u_2[k]), \tag{9}$$

where $u_1[k]$ and $u_2[k]$ are the values of the k-th instance variable for u_1 and u_2. Furthermore, we use the notation '$<$' to denote that condition (9) holds as strict inequality for at least one component $v_k \in \{v_1, \ldots, v_K\}$. It is shown in [41] that if state u_1 is unsafe and state u_2 satisfies $u_1 \leq u_2$, then state u_2 is also unsafe. Hence, under the state representation of (8), the set FB can be effectively defined by the subset of its minimal elements; we shall denote this subset by $\overline{FB}$, i.e., $\overline{FB} \equiv \{u \in FB \mid \nexists u' \in FB \text{ s.t. } u' < u\}$.

The results of our computational experiments that are reported in Section V reveal that the TRIE data structures that provide effective representation of the sets $\overline{FB}$, are considerably more compact, in terms of the employed number of nodes, than the TRIE data structures that encode the original sets FB. Hence, there seems to be substantial advantage in the representation of the forbidden RAS states, during the deployment of the one-step-lookahead scheme of [24], by means of the sets $\overline{FB}$. In the context of this work, we have also developed a symbolic algorithm for the extraction of the set $\overline{FB}$ from its parent set FB; we shall refer to this algorithm as "Algorithm 3" in the sequel. Since, however, the symbolic computation of the minimal elements of a given set of vectors, like that of FB, is a more generic task than the RAS-related concepts and computation that are pursued in this work, we have opted to include this material in the electronic supplement of this paper [37], and we refer the interested reader to that resource for a complete and systematic exposition of the corresponding results.

V. Computational Results and Evaluation

The EFA-based modeling procedure for the considered RAS and the collection of the symbolic algorithms for the computation of their minimal boundary unsafe states that were developed in this work, have been implemented in the DES software tool Supremica [36]. The program is written in Java and it uses JavaBDD [44] with BuDDy as the BDD library. In this section, we report the results from a series of computational experiments[11] in which we applied the symbolic algorithms of Section IV on a number of randomly generated instantiations of the RAS class that was defined in Section II. Each of the generated instances is further specified by:

[11]The experiments were carried out on a standard desktop, (2.66 GHz Intel Core Quad CPU, 10GB RAM) running Windows 7

- The number of resource types in the system; the range of this parameter is between 3 and 16.
- The capacities of the resource types in the system; the range of this parameter is between 1 and 4.
- The number of process types in the system; the range of this parameter is between 3 and 5.
- The number of processing stages in each process; the range of this parameter was between 3 and 16. Furthermore, in order to remain consistent with the RAS structure defined in Section II, no processing stage has a zero resource-allocation vector.
- The structure of the process graphs $\mathcal{G}_i$ and the resource request vectors supported by the resource allocation function $\mathcal{A}$. In terms of the structure of $\mathcal{G}_i$, the generated examples contain RAS instances where all processes follow simple linear flows, as well as RAS instances where some processes possess routing flexibility (Disjunctive RAS). In terms of the resource request vectors, our examples contain RAS instances with either single-type resource allocation or conjunctive resource allocation (Conjunctive RAS).

Table I reports a representative sample of the results obtained in our experiments. The first section in the table is for RAS instances with simple linear process flows and single-unit resource allocation (from a single resource type) for each processing stage. The second section is for RAS instances with simple linear process flows and conjunctive resource allocation for each processing stage. Finally, the last section of the table is for RAS instances with conjunctive resource allocation for each processing stage, but also routing flexibility. In the experiments, first we constructed the necessary BDDs modeling the underlying EFA for the aforementioned RAS instances, then we applied Algorithms 1 and 2 to these BDDs to obtain all the boundary unsafe states, and subsequently we applied Algorithm 3 to remove the non-minimal unsafe states from the set of boundary unsafe states. Finally, we also applied the algorithm of [35] to the BDDs returned by Algorithms 2 and 3 in order to convert these BDDs into the TRIEs that will support the final implementation of the maximally permissive DAP (c.f. the relevant discussion in the previous sections).

Columns 1-2 in Table I report respectively the cardinalities of the set of reachable states, R, and the set of the Boolean variables, $X^{\mathcal{D}}$, that are employed by the BDD $\Delta_{\mathbf{E}}$ which is the primary input to Algorithms 1 and 2. Since, in classical SCT, the maximally permissive DAP is characterized by the trim of the corresponding RAS-modeling FSA, $|R|$ can function as a pertinent surrogate measure of the problem complexity, when it is solved through a conventional representation of this FSA. On the other hand, (7) implies that $|X^{\mathcal{D}}|$ is equal to the number of binary variables that are necessary to encode the resource capacities and (an upper bound to) the maximal number of process instances that can occupy each of the non-terminal processing stages of the considered RAS instance Φ, and therefore, it can be perceived as a surrogate measure of the "size" of Φ in terms of its structural elements (c.f. Definition II.1). Also, in the world of symbolic computation, the number of the Boolean variables that is communicated by $|X^{\mathcal{D}}|$, is frequently used as a practical surrogate measure of

TABLE I: A sample of computational results regarding the efficiency of the presented algorithms

		Algorithm 1 presented in Section IV-C			Algorithm 2 presented in Section IV-D				Algorithm 3 presented in Section IV-E				
$\|R\|$	$\|X^{\mathcal{D}}\|$	$\|RB\|$	T_{RB}	ζ_{RB}	$\|FD\|$	$\|FB\|$	T_{FB}	ζ_{FB}	$\|\overline{FB}\|$	$T_{TRIE}^{\overline{FB}}$	$\zeta_{\overline{FB}}$	$CR(\%)$	$T_{\overline{RU}}$ [25]
799,071	45	186,500	21	527,040	291,592	283,962	7	283,962	8,934	1	551,210	30.77	199
1,659,342	51	381,846	90	1,115,932	942,254	800,940	42	796,123	17,931	7	2,046,115	15.72	1,447
1,962,454	49	438,521	28	607,812	769,090	761,399	29	450,040	10,527	4	929,955	25.51	989
3,436,211	55	783,794	207	1,374,268	1,590,736	1,564,991	106	1,176,110	55,553	73	8,985,355	23.24	5,863
14,158,338	51	2,615,904	180	1,731,691	1,983,934	3,558,362	152	1,561,971	46,048	19	2,939,342	20.38	31,933
14,521,572	54	3,218,012	626	3,556,004	3,399,416	5,696,085	642	4,999,572	51,069	103	7,920,050	1.43	36,305
14,963,458	59	3,207,511	470	3,207,511	6,898,234	5,989,367	553	4,415,000	31,376	393	17,578,012	16.48	51,295
22,212,582	55	5,066,271	2,150	8,019,401	4,621,662	8,056,766	964	5,546,176	62,996	239	12,958,398	16.49	44,222
29,160,898	53	5,496,694	184	2,126,861	3,035,820	7,751,451	237	2,685,162	27,138	25	2,442,037	15.74	14,851
32,380,375	56	8,277,582	609	4,347,910	4,807,088	14,320,225	904	5,415,820	40,306	109	4,506,280	12.65	23,382
2,430,581	55	547,612	15	505,697	247,195	741,764	10	226,991	16,732	0	104,188	38.44	1,112
2,939,463	65	408,009	128	1,432,070	142,301	531,238	97	1,043,925	5,464	0	80,869	25.74	193
1,712,672	57	306,585	83	1,590,899	441,376	445,092	38	646,998	9,563	3	318,205	22.82	1,782
1,962,454	49	438,521	33	613,613	769,080	761,399	25	649,984	10,527	2	996,436	25.33	1,073
6,051,299	54	1,087,093	134	1,547,620	1,189,993	1,781,191	32	575,720	6,292	0	157,214	22.53	1,865
22,212,582	55	5,066,271	946	6,665,791	4,621,662	8,056,766	815	5,182,290	62,996	245	13,028,200	1.63	45,550
24,430,444	64	5,457,497	205	2,408,072	1,037,721	6,000,747	125	1,534,599	10,699	0	228,345	26.46	5,339
29,160,898	53	5,496,694	211	1,898,949	3,035,820	7,751,451	193	2,146,384	27,138	12	1,535,306	15.60	12,649
106,509,798	74	10,910,823	234	3,873,700	841,940	12,529,669	313	2,367,893	4,368	0	43,553	30.23	92
596,212,152	67	139,238,562	3,097	6,791,929	2,033,997	169,402,134	520	6,744,437	572	0	50,278	8.57	1,098
1,663,534	49	130,825	1	175,736	185,177	262,514	1	129,084	6,189	0	30,140	37.32	173
2,340,408	50	342,098	12	525,047	114,926	603,701	2	230,807	2,283	0	13,289	32.49	37
7,885,856	54	425,741	28	1,199,596	383,129	594,828	1	262,861	2,628	0	11,999	32.32	19
30,397,584	53	568,889	24	3,544,487	349,953	853,537	3	229,892	22,318	0	39,358	44.39	1001
81,285,120	76	2,027,904	2	314,728	110,656	4,676,480	0	120,387	1,245	0	1,860	36.46	15
96,438,720	64	5,401,790	365	3,031,243	1,648,506	6,321,838	106	2,526,813	31	0	6,592	7.00	747
399,477,600	92	45,541,152	2,802	14,511,314	2,027,551	122,636,544	59	2,939,165	1,975	0	20,899	24.08	6
1,219,947,240	88	18,531,807	4,987	49,835,897	86,535	72,055,380	460	7,959,586	516	0	495,417	4.22	77
3,547,065,654	94	41,135,520	3,892	20,605,813	1,812,728	93,980,859	74	3,595,817	8,117	0	92,971	11.41	665
3,749,923,584	87	222,163,176	8,773	38,249,085	4,177,807	269,219,724	99	2,441,987	5,171	0	189,467	29.68	88

the overall complexity that is expected for this computation.

Columns 3-5 report on the outcome of the computation of Algorithm 1 and its empirical complexity. More specifically, Column 3 reports the number of the reachable boundary unsafe states, $|RB|$, that are computed by Algorithm 1. Columns 4-5 report the required computation time, T_{RB} (in seconds), and the maximal number of BDD nodes, ζ_{RB}, employed during the execution of the algorithm. Each of these numbers accounts, respectively, for the time and the maximal nodal requirements that are necessary for the construction of the BDD $\Delta_{\mathbf{E}}$, that is a primary input for Algorithm 1, and the considered experimentation has also revealed that these additional quantities are rather insignificant compared to the time and the maximal nodal requirements that characterize the execution of Algorithm 1 itself.

The related results for Algorithm 2 are as follows: Columns 6-7 report the cardinalities of the set of deadlock states FD and the set of boundary unsafe states FB, while Columns 8-9 report the computation time T_{FB} and the peak of the BDD nodes during the execution of this algorithm. Similar to the case of Algorithm 1, both of the last two values account also for the generation of the input BDDs $\Delta_{\mathbf{E}}$ and χ_F.

Columns 10-12 report the results and the computational performance of the application of Algorithm 3, that was discussed in Section IV-E, to the output of Algorithm 2 for the extraction of the minimal elements of that set. This computation also involves the conversion of the obtained BDD to the TRIE data structure that will be used in the final implementation of the sought supervisor. More specifically, Column 10 reports the cardinality of the obtained set of the minimal boundary unsafe states, $\overline{FB}$. Column 11 shows the combined time for the extraction of the minimal boundary unsafe states and the conversion of this set to the aforementioned TRIE construct, $T_{TRIE}^{\overline{FB}}$. On the other hand, Column 12 shows the maximal number of BDD nodes, $\zeta_{\overline{FB}}$, used by Algorithm 3 to remove the non-minimal states from FB.

Column 13 in Table I, labelled by 'CR', expresses the ratio of the number of nodes that are used by the TRIE data structure that stores the elements of the set $\overline{FB}$, that is returned by Algorithm 3, versus the number of nodes of the TRIE data structure that stores the elements of the original set FB that is returned by Algorithm 2.[12] In all our experiments, including those not reported in Table I, this ratio has been substantially smaller than one, and therefore, it is reported as a percentage in Column 13. The aforementioned finding also suggests that there is significant practical advantage, at least in terms of the required memory, in the employment of the set $\overline{FB}$ in the final implementation of the target supervisor.

Finally, Column 14 reports the required computation time for applying the algorithm of [25] on the considered RAS instances. Hence, this column facilitates a performance comparison between that algorithm and the algorithms that have been developed in this work.

Thanks to the compactness offered by the employed symbolic representations, both, Algorithm 1 and Algorithm 2 are capable of handling RAS instances that have billions of states

[12]More accurately, the TRIE data structure encoding the elements of the set FB is constructed after applying to this set the existential quantification operation of Eq. 8. In this way, the compared TRIEs represent vector sets of equal dimensionality.

in their underlying state-spaces, and they manage to compute the set of boundary unsafe states with limited memory and time. Table I also reveals that functions like the removal of non-minimal boundary unsafe states from the originally computed sets RB and FB can be performed very efficiently through symbolic computation. On the other hand, it is evident from Table I that neither $|R|$ nor $|X^{\mathcal{D}}|$ manage to capture effectively the empirical time and space complexity of the computation of the target sets RB and FB respectively by Algorithms 1 and 2, since there is not strong correlation among each of these two columns and the computation time and the maximal BDD-node requirements reported for each of these two algorithms.

Next, we focus on the comparison of the computation time and the maximal memory usage between Algorithm 1 and Algorithm 2. In general, the empirical computational complexity of a symbolic search-based algorithm is mostly dependent on (i) the number of the required iterations for the search process and (ii) the maximum number of BDD nodes employed during the execution of the algorithm. By taking advantage of the particular structure and properties of the considered RAS state spaces, Algorithm 2 avoids the full exploration of these state-spaces. Hence, compared to Algorithm 1 that employs the conventional "trimming" technique for computing the target unsafe states, Algorithm 2 requires fewer iterations to compute the target boundary unsafe states, and it tends to have a better computation time. Furthermore, the avoidance of the exploration of the whole RAS state-space enables Algorithm 2 to consume less memory during its execution, especially for RAS instances with small unsafe state regions. As depicted in Figures 4 and 5, Algorithm 2 outperforms Algorithm 1, on average. This performance dominance is more emphatic for RAS instances with routing flexibility, since, for these RAS, the cardinality of the set of reachable states is orders of magnitude larger than that of the set of boundary unsafe states; therefore, the partial state-space exploration that is effected by Alghorithm 2 establishes a stronger competitive advantage.

We conclude this section by comparing the total computation time of the combined execution of Algorithms 2 and 3 to the computation time of the algorithm for the computation of the minimal boundary unsafe states that is presented in [25]. As demonstrated in Fig. 6, for the RAS instances with simple linear process flows and simple or conjunctive resource allocation, the sequential execution of Algorithms 2 and 3 outperforms the algorithm of [25] in terms of the computation time. Some of the largest cases suggest that the gains attained by the symbolic algorithms can be up to 100 times faster. On the other hand, for RAS instances possessing routing flexibility, the algorithm of [25] is competitive to the symbolic algorithms that were developed herein. We believe that this comparative improvement of the computational efficiency of the algorithm of [25] for these particular RAS instances stems from the fact that the algorithm of [25] focuses explicitly upon *minimal* deadlocks and unsafe states in its computation, and therefore, it effects an even more limited search in the underlying RAS state space compared to the algorithms that are developed herein. The computational gains of this additional restriction of the performed search become more obvious as the underlying state spaces become larger (which is the case with the considered RAS involving routing flexibility).

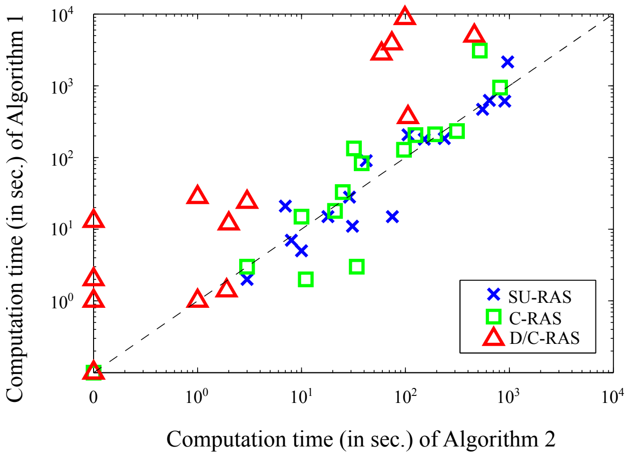

Fig. 4: Comparing the computation times (in sec.) of Algorithms 1 and 2

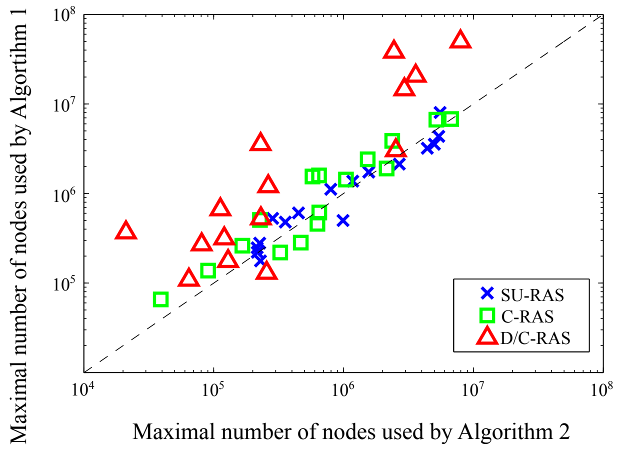

Fig. 5: Comparing the maximal memory usage of Algorithms 1 and 2, based on their maximal BDD-node requirements

VI. Conclusions

This paper has developed a novel methodology for deploying the maximally permissive DAP of any RAS instance coming from the class of D/C-RAS, while taking advantage of the representational and computational efficiencies that are provided by symbolic computation. More specifically, for any given RAS instance, the proposed approach first recasts the underlying resource allocation dynamics into an EFA model, and subsequently, it identifies the feasible boundary unsafe states of this EFA, using one of the two alternative symbolic algorithms that were presented in Section IV. The obtained state set is post-processed by an additional symbolic algorithm that removes all its non-minimal elements, and the resulting BDD can be further converted into a TRIE data structure enabling the implementation of the maximally permissive DAP through the one-step-lookahead scheme for boundary unsafe states presented in [24]. The entire procedure has been implemented in Supremica and a series of computational

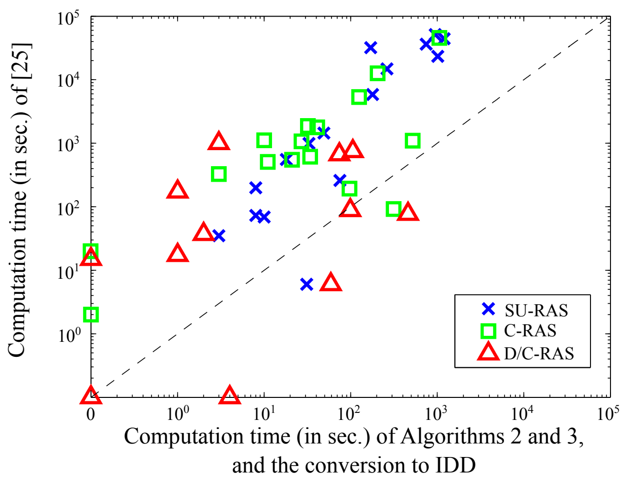

Fig. 6: Comparing the computation time (in sec.) of the sequential execution of Algorithms 2, 3 and the algorithm of [35] to the computation time of the algorithm of [25]

experiments has manifested its efficacy and its computational power.

Since Algorithm 1 relies only on the notion of RAS state safety as defined by the notion of co-accessibility to the RAS empty state, it will work on any RAS instance with fully controllable resource allocation dynamics, beyond the class of D/C-RAS, provided that the employed RAS-modeling EFA is an adequate representation of these dynamics. On the other hand, an additional requirement for the effective application of Algorithm 2 on RAS instances coming from broader RAS classes, is that, in the underlying RAS dynamics, state unsafety still implies the unavoidable absorption of these dynamics to some deadlock state.[13] The two algorithms can also be easily extended to account for uncontrollable RAS dynamics, where uncontrollability is defined either in terms of the timing of some process-loading and advancing events, or in terms of the routing decisions that are effected by the RAS processes at stages possessing routing flexibility. The necessary modifications for Algorithm 1 are in line with the broader logic that underlies the computation of the maximal nonblocking controllable sublanguage in SCT [12], [13]. On the other hand, the aforementioned uncontrollable behavior can be accommodated in the computation of Algorithm 2 by revising the computation of the sets χ_{NU}, $\chi_{U_{cur}}$ and $\chi_{U_{new}}$ in Lines 10-13, to account for the new notion of state unsafety that is implied by the uncontrollable behavior; we leave the relevant details to the reader. In this broader RAS context, the "thinning" of the results of Algorithms 1 and 2 to the corresponding minimal subsets through the application of Algortihm 3 will be possible only if the monotonicity of the state unsafety that was discussed in Section IV-E, extends to this new RAS class; otherwise, the implementation of the target DAP must employ the entire state sets that are returned by these two algorithms, a fact that defines additional advantage for the symbolic representation and computation that is pursued in this work. Finally, [38] also provides an implementation of Algorithm 2 that employs a distributed representation of the characteristic function $\Delta_{\mathbf{E}}$, in an effort to control further the size of the employed BDDs, and thus, the maximal memory footprint of the algorithm. The obtained results reveal that the attained gains can be substantial.

In our future work, we shall consider the possibility of capturing and exploiting, in the effected computation, additional structure that might be present in the underlying RAS state space. Furthermore, it is interesting to investigate the possibility of extending the applicability of the presented symbolic algorithms to RAS with infinite state spaces, like those considered in [46], [45].

[13] A RAS class that might fail to meet this last condition is that containing process types with internal "cycling" in their sequential logic; in these cases, state unsafety might be manifested by eventual absorption to a *"livelock"*, i.e., an entire closed communicating class of non-coreachable states. To the best of our knowledge, currently there are no efficient algorithms for the detection or the programmatic construction of livelocks, and therefore, Algorithm 2 will not be able to execute efficiently the first phase of its overall computation. On the other hand, some important RAS sub-classes that support internal cycling for their process types and, yet, state unsafety implies eventual absorption to a deadlock (and not to a livelock), can be found in [10], [45].

ACKNOWLEDGEMENT

The authors would like to thank Dr. Ahmed Nazeem for providing some of the data that were utilized in the presented computational experiments. Also, the first and the third author want to acknowledge the constructive discussions that they had with Dr. Nazeem while visiting Georgia Tech in Spring 2012. Finally, the first author thanks Dr. Martin Fabian for the technical discussion that he provided on the topics of EFA-based modeling and BDD compression.

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, *Deadlock Resolution in Computer-Integrated Systems*. Singapore: Marcel Dekker, Inc., 2004.
[3] J. Ezpeleta, J. M. Colom, and J. Martinez, "A Petri net based deadlock prevention policy for flexible manufacturing systems," *IEEE Trans. on R&A*, vol. 11, pp. 173–184, 1995.
[4] S. A. Reveliotis and P. M. Ferreira, "Deadlock avoidance policies for automated manufacturing cells," *IEEE Trans. on Robotics & Automation*, vol. 12, pp. 845–857, 1996.
[5] M. P. Fanti, B. Maione, S. Mascolo, and B. Turchiano, "Event-based feedback control for deadlock avoidance in flexible production systems," *IEEE Trans. on Robotics and Automation*, vol. 13, pp. 347–363, 1997.
[6] S. A. Reveliotis, "Conflict resolution in AGV systems," *IIE Trans.*, vol. 32(7), pp. 647–659, 2000.
[7] N. Wu and M. Zhou, "Resource-oriented Petri nets in deadlock avoidance of AGV systems," in *Proceedings of the ICRA'01*. IEEE, 2001, pp. 64–69.
[8] S. Reveliotis and E. Roszkowska, "Conflict resolution in free-ranging multi-vehicle systems: A resource allocation paradigm," *IEEE Trans. on Robotics*, vol. 27, pp. 283–296, 2011.
[9] A. Giua, M. P. Fanti, and C. Seatzu, "Monitor design for colored Petri nets: an application to deadlock prevention in railway networks," *Control Engineering Practice*, vol. 10, pp. 1231–1247, 2006.
[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] H. Liao, Y. Wang, J. Stanley, S. Lafortune, S. Reveliotis, T. Kelly, and S. Mahlke, "Eliminating concurrency bugs in multithreaded software: A new approach based on discrete-event control," *IEEE Trans. on Control Systems Technology*, vol. 21, pp. 2067–2082, 2013.
[12] P. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.
[13] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.

[14] S. Reveliotis and E. Roszkowska, "On the complexity of maximally permissive deadlock avoidance in multi-vehicle traffic systems," *IEEE Trans. on Automatic Control*, vol. 55, pp. 1646–1651, 2010.

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

[16] J. Park and S. A. Reveliotis, "Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings," *IEEE Trans. on Automatic Control*, vol. 46, pp. 1572–1583, 2001.

[17] Z. W. Li and M. C. Zhou, "Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems," *IEEE Trans. on SMC – Part A*, vol. 34, pp. 38–51, 2004.

[18] H. Liao, S. Lafortune, S. Reveliotis, Y. Wang, and S. Mahlke, "Optimal liveness-enforcing control of a class of Petri nets arising in multithreaded software," *IEEE Trans. Autom. Control*, vol. 58, pp. 1123–1138, 2013.

[19] A. Ghaffari, N. Rezg, and X. Xie, "Design of a live and maximally permissive Petri net controller using the theory of regions," *IEEE Trans. on Robotics & Automation*, vol. 19, pp. 137–141, 2003.

[20] E. Badouel and P. Darondeau, "Theory of regions," in *LNCS 1491 – Advances in Petri Nets: Basic Models*, W. Reisig and G. Rozenberg, Eds. Springer-Verlag, 1998, pp. 529–586.

[21] R. Cordone and L. Piroddi, "Monitor optimzation in Petri net control," in *Proceedings of the 7th IEEE Conf. on Automation Science and Engineering*. IEEE, 2011, pp. 413–418.

[22] Y. F. Chen and Z. W. Li, "Design of a maximally permissive liveness-enforcing supervisor with a compressed supervisory structure for flexible manufacturing systems," *Automatica*, vol. 47, pp. 1028–1034, 2011.

[23] A. Nazeem and S. Reveliotis, "Designing maximally permissive deadlock avoidance policies for sequential resource allocation systems through classification theory: the non-linear case," *IEEE Trans. on Automatic Control*, vol. 57, pp. 1670–1684, 2012.

[24] ——, "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.

[25] ——, "Efficient enumeration of minimal unsafe states in complex resource allocation systems," *IEEE Trans. on Automation Science and Engineering*, vol. 11, no. 1, pp. 111–124, 2014.

[26] S. B. Akers, "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol. 27, pp. 509–516, Jun. 1978.

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

[28] G. Hoffmann and H. Wong-Toi, "Symbolic synthesis of supervisory controllers," in *American Control Conference, 1992*, June 1992, pp. 2789–2793.

[29] R. Song and R. Leduc, "Symbolic synthesis and verification of hierarchical interface-based supervisory control," in *Discrete Event Systems, 2006 8th International Workshop on*, July 2006, pp. 419–426.

[30] C. Ma and W. M. Wonham, "Nonblocking supervisory control of state tree structures," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 782–793, May 2006.

[31] A. Vahidi, M. Fabian, and B. Lennartson, "Efficient supervisory synthesis of large systems," *Control Engineering Practice*, vol. 14, no. 10, pp. 1157–1167, Oct. 2006.

[32] Y. Chen, Z. Li, M. Khalgui, and O. Mosbahi, "Design of a Maximally Permissive Liveness-Enforcing Petri Net Supervisor for Flexible Manufacturing Systems," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 2, pp. 374–393, 2011.

[33] 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.

[34] 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.

[35] 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.

[36] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica - An integrated environment for verification, synthesis and simulation of discrete event systems," in *the 8th International Workshop on Discrete Event Systems*, 2006, pp. 384–385.

[37] Z. Fei, S. Reveliotis, S. Miremadi, and K. Åkesson, "Supplement for the paper entitled "A BDD-Based Approach for Designing Maximally Permissive Deadlock Avoidance Policies for Complex Resource Allocation Systems"," Chalmers University of Technology, Tech. Rep., 2014. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/198974/local_198974.pdf

[38] Z. Fei, "Symbolic supervisory control of resource allocation systems," Ph.D. dissertation, Chalmers University of Technology, Gothenburg, Sweden, 2014.

[39] Z. Fei, S. Miremadi, and K. Åkesson, "Modeling sequential resource allocation systems using extended finite automata," in *7th Annual IEEE Conference on Automation Science and Engineering, CASE'11*, Trieste, 2011, pp. 444–449.

[40] Z. Fei, S. Reveliotis, and K. Åkesson, "A symbolic approach for maximally permissive deadlock avoidance in complex resource allocation systems," in *Proceedings of the 12th International Workshop on Discrete Event Systems*. IFAC–IEEE, 2014, pp. 362–369.

[41] 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.

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

[43] C. Baier, J. P. Katoen, and K. G. Larsen, *Principles of Model Checking*. Cambridge, MA: The MIT Press, 2008.

[44] "JavaBDD." [Online]. Available: javabdd.sourceforge.net

[45] A. Nazeem and S. Reveliotis, "Maximally permissive deadlock avoidance for resource allocation systems with R/W-locks," *Discrete Event Systems: Theory and Applications (to appear)*.

[46] ——, "Maximally permissive deadlock avoidance for resource allocation systems with R/W-locks," in *Proceedings of WODES 2012*. IFAC, 2012.

Zhennan Fei received the M.Sc. degree in computer science and the Ph.D. degree in automation from Chalmers University of Technology, Gothenburg, Sweden, in 2009 and 2014, respectively. His research interests include supervisory control of discrete event systems and development and application of formal methods.

Spyros Reveliotis is a Professor in the School of Industrial & Systems Engineering, at the Georgia Institute of Technology. He holds a Diploma in Electrical Engineering from the National Technical University of Athens, Greece, an M.Sc. degree in Computer Systems Engineering from Northeastern University, Boston, and a Ph.D. degree in Industrial Engineering from the University of Illinois at Urbana-Champaign. Dr. Reveliotis' research interests are in the area of Discrete Event Systems theory and its applications. He is a Senior member of IEEE and a member of INFORMS.

Dr. Reveliotis currently serves as a Senior Editor for IEEE Trans. on Automation Science and Engineering, and as Department Editor for IIE Transactions. He has also been a Senior Editor in the Conference Editorial Board for the IEEE Intl. Conference on Robotics & Automation, and an Associate Editor for the IEEE Trans. on Automatic Control, the IEEE Trans. on Robotics & Automation, and the IEEE Trans. on Automation Science and Engineering. In 2009 he was the Program Chair for the IEEE Conference on Automation Science and Engineering. Dr. Reveliotis has been the recipient of a number of awards, including the 1998 IEEE Intl. Conf. on Robotics & Automation Kayamori Best Paper Award.

Sajed Miremadi received the B.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2006; the M.Sc. degree in computer science from Linköping University, Linköping, Sweden, in 2008; and the Ph.D. degree in automation at Chalmers University of Technology, Gothenburg, Sweden, in 2012. His main research interests include supervisory control and optimization of untimed and timed discrete event systems, using formal methods.

Knut Åkesson received the M.S. degree in 1997 in computer science and engineering from Lund Institute of Technology, Lund, Sweden, and the Ph.D. degree in 2002 in control engineering from Chalmers University of Technology, Gothenburg, Sweden. Currently, he is an Associate Professor at the Department of Signals and Systems, Chalmers University of Technology, where his main research interest is the development and application of formal methods to the verification and synthesis of control logic.