Symbolic Computation of Boundary Unsafe States in Complex Resource Allocation Systems Using Partitioning Techniques

Zhennan Fei and Knut Åkesson
Department of Signals and Systems
Chalmers University of Technology
{zhennan, knut}@chalmers.se

Spyros Reveliotis
School of Industrial & Systems Engineering
Georgia Institute of Technology
spyros@isye.gatech.edu

***Abstract*— In some recent work, we proposed a binary decision diagram (BDD-) based approach for the development of the maximally permissive deadlock avoidance policy (DAP) for complex resource allocation systems (RAS), that is based on the identification and the explicit storage of a set of critical states in the underlying RAS state-space. The work presented in this paper seeks to extend the applicability of the aforementioned results by coupling them with a partitioning technique for the more efficient storage and processing of the BDD that encodes the underlying state space. The reported numerical experimentation demonstrates the increased efficiency of the new algorithm w.r.t. its space and time complexity, compared to the previous method that uses a more monolithic representation of the RAS state-space. The last part of the paper also discusses some further potential advantages of the presented method, including its amenability to a parallelized implementation and its ability to cope effectively and efficiently with uncontrollable behavior.**

I. Introduction

The problem of deadlock avoidance in complex, sequential resource allocation systems (RAS) is a well established problem in the controls community [1], [2]. In its basic definition, this problem concerns the (real-time) allocation of a finite set of reusable resources to a set of concurrently running processes that execute in a staged manner, so that no subset of these processes is entangled in a circular waiting pattern for resources currently held by some other process in the set; a resource allocation state containing such a circular waiting pattern among the running processes defines a notion of *deadlock* in the relevant literature [1].

Ideally, one would like to prevent the formation of deadlock while imposing the minimum possible restriction to the underlying resource allocation function and to the concurrent process dynamics that are induced by this function. The corresponding optimal control problem can be effectively modeled in the context of the broader area of Supervisory Control Theory (SCT) for Discrete Event Systems (DES) [1], [3]. The maximally permissive supervisor – also known as the maximally permissive deadlock avoidance policy (DAP) – essentially acts as a classifier that recognizes and blocks transitions to states that are reachable from the initial empty state of the considered RAS but are not co-reachable to that same state. States that are co-reachable to the target empty state are characterized as “safe” in the relevant terminology, while the remaining states are characterized as “unsafe”. Among the unsafe states, particularly interesting to an effective implementation of the maximally permissive DAP are those states that are accessible from a safe state by a single transition, since the recognition and blockage of the transitions leading into these states will render inaccessible every other unsafe state; in the sequel, we shall refer to this particular subclass of unsafe states as “boundary” unsafe states.

Motivated by the above remarks, in [4], we have developed a binary decision diagram (BDD-) based approach for an efficient computation and storage of all the boundary unsafe states. Extensive computational experimentation that is presented in [4] demonstrates the capability of that approach to cope with RAS instances exhibiting very complex behavior and possessing very large state spaces. In this work we seek to extent further the boundary for the applicability of the approach of [4] on the considered supervisory control problem, by introducing a variation for one of the main algorithms of [4] that employs a more efficient representation of the RAS state space. More specifically, we propose to store and process the information that is encoded in this mathematical entity through a partitioning scheme that uses a number of BDDs to store various parts of the involved dynamics. Such a partitioned representation of the RAS dynamics can better manage the state space explosion that is a challenging problem for the monolithic representation of the corresponding state spaces, and as it will be revealed in the following, it leads to an algorithm with a substantially smaller memory footprint than the corresponding algorithm that is presented in [4]. Furthermore, some additional advantages of the proposed method concern its amenability to a parallelized implementation and its ability to cope effectively and efficiently with uncontrollable behavior; we outline these advantages in the closing part of the paper.

Closing this introductory discussion on the paper developments, we must also notice that the employment of pertinent partitioning schemes in BDD-based symbolic computation, in an effort to control the size of the employed BDDs and the ensuing computational complexity, is a well recognized idea that dates back to the seminal work of [5]. Some additional works that have employed such partitioning techniques, including some that develop in the context of SCT, are those reported in [6], [7], [8]. The current work is primarily influenced by relevant results that are presented in [9], and it seeks to apply and customize the more generic theory of [9] to the specific problem that is addressed herein.

II. Preliminaries

We beginning the technical discussion of this paper by providing the necessary background for the presentation of the novel method that is proposed in this work; the method

itself is presented in Section III. Due to space limitations, we keep the provided discussion at a minimal level; a more expansive treatment of this material can be found in [4], [10].

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

For the purposes of this work, a *resource allocation system (RAS)* is defined by a 4-tuple $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ where:[1] (i) $\mathcal{R} = \{R_1, \ldots, R_m\}$ is the set of the system *resource types*. (ii) $C : \mathcal{R} \rightarrow \mathbb{Z}^+$ – 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. (iii) $\mathcal{P} = \{J_1, \ldots, J_n\}$ 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. (iv) $\mathcal{A} : \bigcup_{j=1}^{n} \mathcal{S}_j \rightarrow \prod_{i=1}^{m}\{0, \ldots, C_i\}$ 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-the 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.[2]

The "hold-while-waiting" protocol that is described above, 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. As remarked in the introductory section, 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 its behavior. The relevant control problem is known as *deadlock avoidance*, and, as mentioned in the introductory section, a natural framework for its investigation is that of DES Supervisory Control Theory (SCT) [3]. More specifically, in a finite state automaton (FSA)-based representation of the RAS dynamics, the initial and the target – or, more formally, the "marked" – system state correspond to the state where the RAS is idle and empty of any processes, and deadlock avoidance translates to the development of the *maximally permissive non-blocking supervisor* for this RAS-modeling FSA. The latter confines 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 idle and empty state. Furthermore, 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*. Of particular interest in the implementation of the maximally permissive non-blocking supervisor for the considered RAS are those transitions leading from safe to 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.

A state-of-the-art approach for the computation of all the boundary unsafe states of any RAS instance in the class of D/C-RAS is provided in [4]. More specifically, the approach introduced in [4] first recasts the resource allocation dynamics of the considered RAS instance into an *extended finite automaton* (EFA) [11], which is further encoded in a *binary decision diagram* (BDD) [12], [13]. The main algorithm of [4] that is of interest in this work subsequently employs this BDD, as well some structural characterizations regarding the RAS state safety from [14], in order to compute the underlying set of boundary unsafe states through a two-stage computation. In the first stage, all the deadlock states are identified and retrieved from the aforementioned BDD-based representation of the RAS state-space. In the second stage, the identified deadlock states are used as starting points for a search procedure over the aforementioned BDD that identifies all the boundary unsafe states. The work presented in this paper seeks to provide a more efficient version of this algorithm that is based on a more distributed representation of the BDD that models the underlying RAS state space. But in order to proceed with the formal statement and analysis of this new algorithm, we need to provide some further background on the EFA-based modeling of the considered RAS and the BDD-based modeling of these EFA.

B. Modeling the Considered RAS as Extended Finite Automata

The Extended Finite Automaton (EFA) [11] 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. More formally, 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 (i) $Q : L \times \mathcal{D}$ is the extended finite set of states. L is the finite set of the model *locations* and $\mathcal{D} = \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$ is the finite domain of the model *variables* $v = (v_1, \ldots, v_n)$. (ii) Σ is a nonempty finite set of events. (iii) $\rightarrow \subseteq L \times \Sigma \times G \times A \times L$ is the

[1]The considered RAS class is known as the class of Disjunctive/Conjunctive (D/C-) RAS in the literature, since it enables routing flexibility for its process types and requests for arbitrary resource sets at the various processing stages [1].

[2]We remind the reader that $x^+ = \max\{0, x\}$ and $x^- = \min\{0, x\}$. When these expressions are applied on vectors, as is the case in the considered text, their application is meant to be componentwise.

transition relation, describing a set of transitions that take place among the model locations upon the occurrence of certain events. However, these transitions are further qualified by G, which is a set of guard predicates defined on $\mathcal{D}$, and by A, which is a collection of actions that update the model variables as a consequence of an occurring transition. Each action $a \in A$ is an n-tuple of functions $(a_1, \ldots, a_n)$, with each function a_i updating the corresponding variable v_i. (iv) $s_0 = (\ell_0, v_0) \in L \times \mathcal{D}$ is the initial state, where ℓ_0 is the initial location, while v_0 denotes the vector of the initial values for the model variables. (v) $Q^m \subseteq L^m \times \mathcal{D}^m \subseteq Q$ is the set of marked states. $L^m \subseteq L$ is the set of the marked locations and $\mathcal{D}^m \subseteq \mathcal{D}$ denotes the set of the vectors of marked values for the model variables.

EFA-based modeling of RAS dynamics The formal construction of an EFA $E(\Phi)$ modeling the dynamics of any given RAS instance $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ is presented in [4]. For the needs of this manuscript, this construction is briefly illustrated in the following example.

Example II.1: The RAS instance Φ under consideration is shown in Fig. 1. It comprises two process types J_1 and J_2, each of which is defined as a sequence of three processing stages; the stages of process type $J_j,\ j = 1, 2$, are denoted by $\Xi_{jk},\ k = 1, 2, 3$. The system resource set is $\mathcal{R} = \{R_1, R_2, R_3\}$, with capacity $C_i = 1$ for $i = 1, 2, 3$. Each processing stage Ξ_{jk} requests only one unit from a single resource type; the relevant resource allocation function is depicted in Fig. 1.

Fig. 2 shows the EFA that models the behavior of process J_1 in the RAS of Fig. 1. This EFA has only one location, and its three transitions correspond to the loading and the process-advancing events among its different stages. More specifically, in the EFA depicted in Fig. 2, the evolution of a process instance through the various processing stages is traced by the *instance variables* $v_{1k},\ k = 1, 2$; each of these variables counts the number of process instances that are executing the corresponding processing stage. The model does not avail of a variable v_{13} since it is assumed that a process instance reaching stage Ξ_{13} is (eventually) unloaded from the system, without the need for any further resource allocation action.[3]

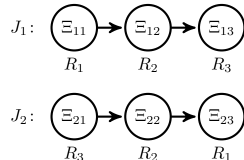

Fig. 1: A simple RAS

The aforementioned EFA $E(\Phi)$ that model the process types J_1 and J_2 are linked through the global *resource variables* $vR_i,\ i = 1, 2, 3$, where each variable vR_i denotes the number of free units of resource R_i. Hence, the domain of variable vR_i is $\{0, \ldots, C_i\}$. Since, under proper RAS operation, the initial and the target final state correspond to the empty state, both the initial and the marked values of each variable vR_i are equal to C_i, and the corresponding values for all instance variables v_{jk} are equal to zero.

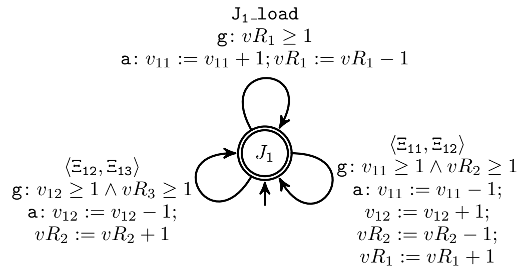

Fig. 2: The resource-augmented EFA for J_1

Finally, as depicted in Fig. 2, the resource and the instance variables are used to construct the necessary guards and actions for the system transitions. The guards determine whether a process-loading or advancing event can take place, on the basis of the process and the resource availability. Upon the occurrence of such an event, the corresponding actions update accordingly the available resource units and the process instances that are active at the various processing stages.

State feasibility It is evident from the above description 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 constraints

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

where $\mathcal{T}(j)$ denotes the set with all the terminal stages of process type J_j. The constraints of (1) 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 traced RAS dynamics. In the following, any state s of the EFA $E(\Phi)$ with a variable vector v satisfying the constraints of (1) will be characterized as a *feasible* state.

C. Encoding EFA Dynamics by Binary Decision Diagrams

The Binary Decision Diagram (BDD) [13] is a memory-efficient data structure used to represent Boolean functions as well as to perform set-based operations. For any Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ 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:

$$\forall x_i \in X,\ f = (\neg x_i \land f|_{x_i=0}) \lor (x_i \land f|_{x_i=1}) \qquad (2)$$

More specifically, (2) 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

[3] However, we should further clarify that the omission of the terminal stage Ξ_{13} 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.

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 efficiency of BDDs is mainly due to the fact that the worst-case complexity of performing some logical operation on two functions f and f' is $\mathcal{O}(|f| \cdot |f'|)$, where $|f|$ and $|f'|$ are the sizes of the BDDs representing f and f'. A particular operator that is used extensively in the following is the *existential quantification* of a function f over some of its Boolean variables. For a variable $x \in X$, the existential quantification of f over x is defined by $\exists x.f \equiv f|_{x=0} \vee f|_{x=1}$. Also, if $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_k) \subseteq X$, then $\exists \bar{X}.f$ is a shorthand notation for $\exists \bar{x}_1.\exists \bar{x}_2.\ldots \exists \bar{x}_k.f$.

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 $q \notin \bar{Q}$.[4] The symbolic representation of the transition relation $\rightarrow$ relies on the same idea. A transition is essentially a tuple $\langle \ell, v, \sigma, \ell', v' \rangle$ specifying a source state $q = (\ell, v)$, an event σ, and a target state $q' = (\ell', v')$. Formally, we employ the variable sets X^L and $X^{\mathcal{D}}$ to encode the source state q, and a copy of X^L and $X^{\mathcal{D}}$, denoted by $\acute{X}^L$ and $\acute{X}^{\mathcal{D}}$, to encode the target state q'. In addition, we employ the Boolean variable set X^{Σ} to encode the alphabet of E, and we associate the event σ with a unique satisfying assignment of the variables in X^{Σ}. Then, we identify the transition relation $\rightarrow$ of E with the characteristic function

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

That is, Δ assigns 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.

BDD-based modeling of the RAS behavior 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 these EFA. The resource allocation dynamics generated by Φ can be formally expressed

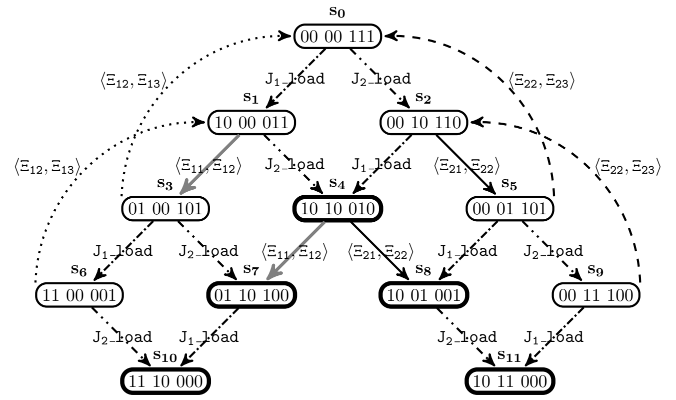

Fig. 3: The state transition diagram (STD) modeling the RAS dynamics of the example depicted in Fig. 1.

by the extended full synchronous composition (EFSC) [11] that composes the aforementioned EFA to the "plant" EFA $\mathbf{E} = E_1 || \ldots || E_n$. In [4], the symbolic representation of the transition relation of $\mathbf{E}$, denoted by $\Delta_{\mathbf{E}}$, is obtained from $\Delta_1, \ldots, \Delta_n$ through the approach of [15], that constructs a single (or a "monolithic") BDD to represent $\Delta_{\mathbf{E}}$. In this work, following the ideas of [9], we propose to employ a more distributed representation of $\Delta_{\mathbf{E}}$ that is based on the construction of a set of disjunctive partial transition relations, denoted by $\{\Delta_\sigma \mid \sigma \in \Sigma_{\mathbf{E}}\}$, which, collectively, will represent $\Delta_{\mathbf{E}}$, i.e., $\Delta_{\mathbf{E}} \triangleq \bigvee_{\sigma \in \Sigma_{\mathbf{E}}} \Delta_\sigma$.

The primary motivation for such a more distributed representation of $\Delta_{\mathbf{E}}$ comes from the following facts: From the previous discussion on BDDs and their corresponding efficiencies, it follows that the expected computational complexity for the execution of some logical operation between BDD $\Delta_{\mathbf{E}}$ and some other BDD χ_B is $O(|\Delta_{\mathbf{E}}| \cdot |\chi_B|)$, if the monolithic representation of $\Delta_{\mathbf{E}}$ is used.[5] On the other hand, this complexity is given by $\sum_{\sigma \in \Sigma_{\mathbf{E}}} O(|\Delta_\sigma| \cdot |\chi_B|)$, if the proposed, more distributed representation of $\Delta_{\mathbf{E}}$ can be employed. Hence, if the BDDs representing each of the partial transition relations Δ_σ, $\sigma \in \Sigma_{\mathbf{E}}$, contain much fewer nodes than the single BDD representing the monolithic transition relation $\Delta_{\mathbf{E}}$, the expected complexity of performing the aforementioned operation on $\Delta_{\mathbf{E}}$ by means of its distributed representation will be substantially lower. Such a complexity reduction can be even more dramatic when comparing the *space* complexities of the two approaches, since, for the distributed case, the corresponding space complexity is given by $\max_{\sigma \in \Sigma_{\mathbf{E}}} \{O(|\Delta_\sigma| \cdot |\chi_B|)\}$.

Closing this discussion on the modeling of the RAS behavior through BDDs, we also notice that, as in [4], the computations pursued in this work do not require the encoding of the locations of the considered EFA, since these elements 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. The realization of this fact enables the further compression of each of the aforementioned BDDs Δ_σ by executing on it the existential quantification $\Delta_\sigma := \exists (X_1^L \cup \ldots \cup X_n^L).\Delta_\sigma$.

Example II.2: Fig. 3 depicts the dynamic behavior of

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

[5]We remind the reader that for a BDD χ_B, $|\chi_B|$ denotes the number of decision nodes of this BDD.

the RAS instance in Example II.1 using the BDD-related concepts that were discussed in the previous paragraph. The depicted state transition diagram (STD) includes only the RAS feasible states that are reachable from the initial state of the corresponding EFA $E(\Phi)$, and furthermore, it considers only those states that are modeled explicitly in this EFA through the pricing of the corresponding model variables. As it can be seen in Fig. 3, the resulting STD involves twelve (12) states, with each state s_i, $i = 0, \ldots, 11$, being described by seven components that correspond to the values of the instance variables $v_{11}, v_{12}, v_{21}, v_{22}$ and the resource variables vR_1, vR_2, vR_3 of the EFA $E(\Phi)$. States depicted with a thick border are unsafe. Also, as mentioned above, we could construct a single monolithic BDD $\Delta_{\mathbf{E}}$ to symbolically represent the set of all transitions in the considered STD. Alternatively, we can construct six simpler BDDs that will represent the BDD $\Delta_{\mathbf{E}}$ in a combined manner; each of these six BDDs collects the transitions corresponding to a particular event recognized by the EFA $E(\Phi)$, and it is indicated by a specific line style in Fig. 3.

III. THE NEW ALGORITHM

In this section, we present our new symbolic algorithm for retrieving the boundary unsafe states of the underlying RAS state-space that is symbolically represented by the BDD set $\{\Delta_\sigma \mid \sigma \in \Sigma_{\mathbf{E}}\}$. In the proposed implementation, we further differentiate these BDDs into two subsets $\{\Delta_{\sigma_1}, \ldots, \Delta_{\sigma_\mu}\}$ and $\{\Delta_{\sigma'_1}, \ldots, \Delta_{\sigma'_k}\}$, with σ_i, $1 \leq i \leq \mu$, and σ'_j, $1 \leq j \leq k$, being the process-advancing and loading events, respectively. Since, for most RAS instances, the number of loading events is rather small, we conveniently group the corresponding BDDs together, by performing the disjunction operation on them. We shall denote the resultant BDD by the characteristic function Δ_L, i.e., $\Delta_L := \Delta_{\sigma'_1} \vee \ldots \vee \Delta_{\sigma'_k}$. Moreover, as it will be revealed in the sequel, the algorithm establishes and maintains the feasibility of the various extracted states by utilizing the characteristic function χ_F that expresses the state-feasibility conditions of (1) in the BDD-based representational context; this function can be systematically constructed by first expressing collectively the invariants of (1) through the Boolean function

$$\bigwedge_{i=1}^{m} \Big(vR_i + \sum_{j=1}^{n} \sum_{k \in \{1,\ldots,l(j)\} \setminus \mathcal{T}(j)} \mathcal{A}_{jk}[i] * v_{jk} = C_i\Big), \quad (3)$$

and subsequently setting χ_F equal to 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 (3).

As in [4], the proposed symbolic algorithm for computing the feasible boundary unsafe states is decomposed into two stages. In the first stage, all the deadlock states w.r.t. the process-advancing events are identified and computed. In the second stage, the deadlock states are used as starting points for a search procedure that identifies all the boundary unsafe states. The entire computation is formally expressed by Algorithm 1, that works with the BDDs representing $\Delta_{\sigma_1}, \ldots, \Delta_{\sigma_\mu}$, Δ_L and χ_F, and returns the characteristic functions χ_{FD} and χ_{FB} that constitute respective symbolic representations of the sets of the feasible deadlock states and

Algorithm 1: SYMBOLIC COMPUTATION OF FB

Input: $\Delta_{\sigma_1}, \ldots, \Delta_{\sigma_\mu},\ \Delta_L$ and χ_F
Output: $\chi_{FD},\ \chi_{FB}$

/ Compute the feasible deadlock states χ_{FD} */*
1 $\chi_{ND} := \chi_{\{s_0\}}$
2 **for** $i \in \{1, \ldots, \mu\}$ **do**
3 $\quad \chi_{ND} := \chi_{ND} \vee (\exists \acute{X}^{\mathcal{D}}.\ \Delta_{\sigma_i})$
4 $\chi_{D_1} := 0$
5 **for** $i \in \{1, \ldots, \mu\}$ **do**
6 $\quad \chi_T^{\sigma_i} := (\exists X^{\mathcal{D}}.\ \Delta_{\sigma_i})[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]$
7 $\quad \chi_{D_1} := \chi_{D_1} \vee (\chi_T^{\sigma_i} \wedge \neg\chi_{ND})$
8 $\chi_{D_2} := ((\exists X^{\mathcal{D}}.\ \Delta_L)[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]) \wedge \neg\chi_{ND}$
9 $\chi_D := (\chi_{D_1} \vee \chi_{D_2})$
10 $\chi_{FD} := \chi_D \wedge \chi_F$

/ Compute the feasible boundary unsafe states from χ_{FD} */*
11 $\chi_{U_{new}} := \chi_{FD},\ \chi_U := \chi_{FD}$
12 $\chi_{LU}^{\sigma_i} := 0,\ \Delta_{U_{pre}}^{\sigma_i} := 0\ \forall i \in \{1, \ldots, \mu\}$
13 **repeat**
14 $\quad$ **for** $i \in \{1, \ldots, \mu\}$ **do**
15 $\quad\quad \Delta_U^{\sigma_i} := (\chi_{U_{new}}[X^{\mathcal{D}} \to \acute{X}^{\mathcal{D}}]) \wedge \Delta_{\sigma_i}$
16 $\quad\quad \chi_{SU}^{\sigma_i} := \exists \acute{X}^{\mathcal{D}}.\ \Delta_U^{\sigma_i}$
17 $\quad\quad \chi_{LU}^{\sigma_i} := \chi_{LU}^{\sigma_i} \vee \chi_{SU}^{\sigma_i}$
18 $\quad\quad \chi_{NU}^{\sigma_i} := (\exists \acute{X}^{\mathcal{D}}.\ \Delta_{\sigma_i}) \wedge \neg\chi_{LU}^{\sigma_i}$
19 $\quad\quad \Delta_{U_{pre}}^{\sigma_i} := \Delta_{U_{pre}}^{\sigma_i} \vee \Delta_U^{\sigma_i}$
20 $\quad \chi_{U_{new}} := 0$
21 $\quad$ **for** $i \in \{1, \ldots, \mu\}$ **do**
22 $\quad\quad \chi_{temp}^{\sigma_i} := \chi_{SU}^{\sigma_i}$
23 $\quad\quad$ **for** $j \leftarrow 1$ **to** μ *s.t.* $i \neq j$ **do**
24 $\quad\quad\quad \chi_{temp}^{\sigma_i} := \chi_{temp}^{\sigma_i} \wedge \neg\chi_{NU}^{\sigma_j}$
25 $\quad\quad \chi_{U_{new}} := \chi_{U_{new}} \vee \chi_{temp}^{\sigma_i}$
26 $\quad$ **for** $i \in \{1, \ldots, \mu\}$ **do**
27 $\quad\quad \Delta_{U_{pre}}^{\sigma_i} := \Delta_{U_{pre}}^{\sigma_i} \wedge \neg\chi_{U_{new}}$
28 $\quad \chi_U := \chi_U \vee \chi_{U_{new}}$
29 **until** $\chi_{U_{new}} = 0$
30 $\Delta_{LU} := (\chi_U[X^{\mathcal{D}} \to \acute{X}^{\mathcal{D}}]) \wedge \Delta_L$
31 $\chi_{FBL} := (\exists X^{\mathcal{D}}.\ (\Delta_{LU} \wedge \neg\chi_U))[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}]$
32 $\chi_{FBA} := 0$
33 **for** $i \in \{1, \ldots, \mu\}$ **do**
34 $\quad \chi_{FBA} := \chi_{FBA} \vee ((\exists X^{\mathcal{D}}.\ \Delta_{U_{pre}}^{\sigma_i})[\acute{X}^{\mathcal{D}} \to X^{\mathcal{D}}])$
35 $\chi_{FB} := \chi_{FBL} \vee \chi_{FBA}$

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 and empty state s_0. However, the presence of these additional states 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 document.[6]

Identification of the feasible deadlock states The symbolic operations for the computation of the characteristic function χ_{FD} are depicted in Lines 1-10 of Algorithm 1, and they can be described as follows:

(i) The first step, consisting of Lines 1-3, computes the characteristic function χ_{ND} for all the non-deadlock states in $\Delta_{\sigma_1}, \ldots, \Delta_{\sigma_\mu}$; these states are identified as states that can enable some process-advancing event. (ii) Subsequently, by utilizing the characteristic function χ_{ND}, Lines 4-9 compute the characteristic function χ_D of all deadlock states. These are states that result from some loading or process-advancing

[6]The reader is referred to [4] for further discussion on the implementation of the target DAP by means of the computed set χ_{FB}.

event and they do not belong in the set of non-deadlock states χ_{ND}. Each of these two deadlock subsets are collected respectively in the sets χ_{D_1} and χ_{D_2}.[7] Line 9 merges symbolically the two identified deadlock sets χ_{D_1} and χ_{D_2} into the characteristic function χ_D, but the resulting state set might contain deadlock states that are infeasible, i.e., they violate the resource-induced invariants of (1). Hence, (iii) in the last step of the first stage of Algorithm 1, 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}.

Identification 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 state set χ_{FB} as follows: (i) At this phase of the computation, Algorithm 1 employs the set U to collect all the identified unsafe states. These unsafe states are obtained through an iterative process, where the state set U_{new} contains the new unsafe states that are obtained at a certain iteration. These states subsequently become the starting points for an one-step-backtracing process in the STDs corresponding to each of the BDDs $\Delta_{\sigma_1}, \ldots, \Delta_{\sigma_\mu}$, in an effort to reach and identify new unsafe states in the next iteration. The corresponding symbolic representations for the two aforementioned sets, denoted by χ_U and $\chi_{U_{new}}$, are initialized to χ_{FD}. Furthermore, for each event σ_i, we define and maintain the sets $LU^{\sigma_i} \equiv \{s \mid (s,u) \in \Delta_{\sigma_i} \wedge u \in U\}$ and $U^{\sigma_i}_{pre} \equiv \{(s,u) \in \Delta_{\sigma_i} \mid u \in U \wedge s \notin U\}$, which are symbolically represented by the respective characteristic functions $\chi^{\sigma_i}_{LU}$ and $\Delta^{\sigma_i}_{U_{pre}}, \forall i = 1, \ldots, \mu$.

(ii) During the main iteration of the executed search process, corresponding to Lines 14-27 in Algorithm 1, the algorithm first performs the following operations w.r.t. each Δ_{σ_i} (Lines 14-18): The algorithm first collects in $\chi^{\sigma_i}_{SU}$ the states that can be reached from the unsafe states in U_{new} through backtracing on the transitions of Δ_{σ_i}, and it also adds this set of states in $\chi^{\sigma_i}_{LU}$. Since states in $\chi^{\sigma_i}_{LU}$ are states with emanating transitions in Δ_{σ_i} that lead to unsafe states, any state in this set could be perceived as "unsafe" if it was evaluated for safety only against the particular transition subset that is encoded by Δ_{σ_i}. The algorithm also computes the set of states $\chi^{\sigma_i}_{NU}$ that have emanating transitions in Δ_{σ_i} and do not belong in $\chi^{\sigma_i}_{LU}$; these are states that currently cannot be pronounced as "unsafe" when assessed from the viewpoint of the transition set encoded by Δ_{σ_i}.

(iii) With the state sets $\chi^{\sigma_i}_{SU}$ and $\chi^{\sigma_i}_{NU}$ available for every $i = 1, \ldots, \mu$, the algorithm proceeds to compute the new unsafe states that are revealed in this iteration (Lines 20-25): The defining logic for this computation is that a state in some of the obtained sets $\chi^{\sigma_i}_{SU}$ will be (really) unsafe, only if it is perceived as "unsafe" w.r.t. every transition set Δ_{σ_i}, i.e., only if it does not belong to any $\chi^{\sigma_j}_{NU}$, for $j \neq i$. States that satisfy this criterion are collected in the set $\chi_{U_{new}}$, which is re-initialized to zero at the beginning of this phase of the overall computation (Line 20) and it is eventually added to the state set χ_U (Line 28).

(iv) During the computational phases described in items (ii) and (iii) above, the algorithm also maintains the transition sets $U^{\sigma_i}_{pre},\ i = 1, \ldots, \mu$, (Lines 19 and 27). At the end of the described iterations, these sets contain the "boundary" transitions between the safe and the unsafe regions w.r.t. each corresponding event σ_i.

(v) The iteration described in items (ii)-(iv) above terminates when no new unsafe sates can be identified by the algorithm. At this point, Algorithm 1 proceeds to extract the boundary unsafe states, and the corresponding operations can be described as follows: First, at Line 30, the algorithm extracts in the characteristic function Δ_{LU} all the loading transitions with their target states in χ_U. Subsequently, Line 31 uses the outcome of Line 30 in order to compute the boundary unsafe states that are reached from safe states through loading events; the set of boundary unsafe states identified at Line 31 is represented by the characteristic function χ_{FBL}. Next, the algorithm continues to identify the rest of the boundary unsafe states that are reached from some safe states through process-advancing events (Lines 32-34). This set of boundary unsafe states is represented by the characteristic function χ_{FBA} and, as explained in item (iv) above, it can be computed by collecting all the target states of the transitions in $\Delta^{\sigma_i}_{U_{pre}}, \forall i = 1, \ldots, \mu$. Finally, χ_{FB} is obtained in Line 35 by taking the disjunction of χ_{FBL} and χ_{FBA}.

Example III.1: As a concrete example, we apply Algorithm 1 to the STD depicted in Fig. 3 to identify its boundary unsafe states. The transitions of the STD are partitioned and symbolically represented in the following five BDDs: Δ_L represents the transitions of the considered STD that are labelled by process loading events (i.e., $\mathtt{J_1_Load}$ and $\mathtt{J_2_Load}$); $\Delta_{\langle \Xi_{11}, \Xi_{12} \rangle}, \Delta_{\langle \Xi_{12}, \Xi_{13} \rangle}, \Delta_{\langle \Xi_{21}, \Xi_{22} \rangle}$ and $\Delta_{\langle \Xi_{22}, \Xi_{23} \rangle}$ represent the transitions that are labeled by the corresponding process-advancing events.

The application of Lines 1-10 of Algorithm 1 to the aforementioned BDDs will return the BDD of feasible deadlock states, χ_{FD}, that includes the states $\mathbf{s_7}, \mathbf{s_8}, \mathbf{s_{10}}$, and $\mathbf{s_{11}}$. Indeed, it can be easily checked in the depicted STD that these states enable no process-advancing events.[8]

Next, starting from the aforementioned deadlock states, the backward search through the RAS process-advancing transitions, implemented in Lines 13-29 of Algorithm 1, will identify all the deadlock-free unsafe states of the considered STD. More specifically, at the first iteration of this search, state $\mathbf{s_4}$ will be reached by backtracing from states $\mathbf{s_7}$ and $\mathbf{s_8}$, with the respective backtracing taking place in BDDs $\Delta_{\langle \Xi_{11}, \Xi_{12} \rangle}$ and $\Delta_{\langle \Xi_{21}, \Xi_{22} \rangle}$. Hence, after the execution of Line 17, state sets $\chi^{\langle \Xi_{11}, \Xi_{12} \rangle}_{LU}$ and $\chi^{\langle \Xi_{21}, \Xi_{22} \rangle}_{LU}$ will contain state $\mathbf{s_4}$ while $\chi^{\langle \Xi_{12}, \Xi_{13} \rangle}_{LU}$ and $\chi^{\langle \Xi_{22}, \Xi_{23} \rangle}_{LU}$ will remain empty. Subsequently, the computation at Line 18 collects, for each BDD $\Delta_\sigma,\ \sigma \in \{\langle \Xi_{11}, \Xi_{12} \rangle, \langle \Xi_{21}, \Xi_{22} \rangle, \langle \Xi_{12}, \Xi_{13} \rangle, \langle \Xi_{22}, \Xi_{23} \rangle\}$, all the states in that Δ_σ with their emanating transitions leading to states that can not be identified as unsafe states at the current iteration; in particular, $\chi^{\langle \Xi_{11}, \Xi_{12} \rangle}_{NU} = \{\mathbf{s_1}\}$, $\chi^{\langle \Xi_{21}, \Xi_{22} \rangle}_{NU} = \{\mathbf{s_2}\}$, $\chi^{\langle \Xi_{12}, \Xi_{13} \rangle}_{NU} = \{\mathbf{s_3}, \mathbf{s_6}\}$ and $\chi^{\langle \Xi_{22}, \Xi_{23} \rangle}_{NU} = \{\mathbf{s_5}, \mathbf{s_9}\}$. Since state $\mathbf{s_4}$ does not belong to any of these four

[7] The operation $[\acute{X}^{\mathcal{D}} \rightarrow X^{\mathcal{D}}]$ denotes the replacement of all variables of $\acute{X}^{\mathcal{D}}$ by those of $X^{\mathcal{D}}$, so that the symbolic computation can proceed.

[8] But they might still enable some loading events, as is the case with states $\mathbf{s_7}$ and $\mathbf{s_8}$.

state sets, it will be identified as an unsafe state through the computations depicted in Lines 21-25. For this example, the backward search terminates after the second iteration, since the attempt to backtrace from state $\mathbf{s_4}$ through some process-advancing transitions fails to reach any states at all. Finally, Algorithm 1 proceeds to extract the boundary unsafe states. Since, in this example, all the unsafe states $\mathbf{s_4}, \mathbf{s_7}, \mathbf{s_8}, \mathbf{s_{10}}$ and $\mathbf{s_{11}}$ can be reached from the safe subspace through a single transition, the resulting state set χ_{FB} contains all these states in the set χ_U. □

A complete correctness analysis of Algorithm 1 that establishes (i) its finite termination and (ii) the soundness of the performed computation, can be found in [10].[9] Next, we proceed to discuss some experimental results that establish the efficacy of this algorithm w.r.t. the corresponding algorithm of [4].

Experimental results The symbolic algorithms for computing boundary unsafe states that was developed in this work, has been implemented in the DES software tool Supremica [17]. The program is written in Java and it uses JavaBDD [18] with BuDDy as the BDD library. Table I reports the experimental results of applying the proposed symbolic algorithm to a set of representative RAS instances.[10] These RAS instances are grouped into three categories: (i) The first category involves RAS with linear process flows and single-unit requests, from a single resource type, by each processing stage. (ii) The second category retains the linear structure of the process flows, but allows for arbitrary resource allocation requests by the various processing stages, in terms of the requested resource types and their quantities. (iii) The third category preserves the arbitrary resource allocation requests of the second category, but it also allows for routing flexibility. Columns 1-2 in Table I report, respectively, the cardinalities of the set of reachable states R and the set of feasible boundary unsafe states FB. Columns 3-4 report the required computation time, t_o (in secs), and the maximal number of BDD nodes, ζ_o, employed during the execution of the symbolic algorithm introduced in [4]. On the other hand, Columns 5-6 report the required computation time, t_n, and the maximal number of BDD nodes, ζ_n, employed during the execution of the algorithm presented in this work. Finally, Columns 7-8 report the relative reduction to the computation time and the maximal number of BDD nodes that is incurred by the new algorithm; a negative value in any of these two columns should be interpreted as an increase of the corresponding quantity.[11]

The perusal of the data shown in Table I reveals that, as expected, (i) Algorithm 1 is more efficient in terms of its memory requirements compared to the symbolic algorithm of [4], for all RAS instances. (ii) The computation times of Algorithm 1 are also improved for most of the RAS instances in the first two categories, even though more operations are needed in Lines 13-28 of Algorithm 1 compared to the algorithm of [4]. (iii) For the RAS instances with routing flexibility, because of the scarcity of the unsafe states in the corresponding state spaces, and the larger "depth" (i.e., the maximal distance from a deadlock state to any boundary unsafe state) of the unsafe state-space, the presented symbolic algorithm requires longer computation time compared to the algorithm of [4]. However, this gap is not large for most of the presented cases, as depicted in Table I.[12] Moreover, as discussed in the next section, the execution time of the considered algorithm can be further controlled through parallelization.

IV. EXTENSIONS

Parallelization Algorithm 1 can be easily parallelized on shared multi-processor or multi-core systems by a series of straightforward modifications. First, regarding the computation of χ_{FD}, we notice that the symbolic operation depicted at Line 3 can be parallelized since the extraction of the source states from the transitions of each $\Delta_{\sigma_i}, \forall i = 1, \ldots, \mu$, can be performed independently on each set. Once χ_{ND} is obtained, it is also evident that the operations of Lines 6-7 can be run in parallel on each Δ_{σ_i}, while at the same time the operation of Line 8 can be executed on Δ_L. Finally, Lines 9-10 will use the outcomes of these partial computations in order to compute χ_{FD}. Similarly, the part of Algorithm 1 for computing χ_{FB} can be parallelized w.r.t. the operations carried out in Lines 14-27 and 33-34; in this parallelizing scheme, the sets $\chi_{LU}^{\sigma_i}$ and $\Delta_{U_{pre}}^{\sigma_i}$ need to be bundled together with Δ_{σ_i} in the corresponding threads. On the other hand, it should also be noticed that the execution of the operations of Lines 22-25 in each of the aforementioned threads necessitates the dissemination among them of the sets $\chi_{NU}^{\sigma_i}$ that are computed by each of them. Hence, there is a need for information exchange among these threads. This information exchange can take place asynchronously among the running threads, and, in certain cases, it might be possible to exploit the particular structure of the underlying RAS in order to reduce further this communication coupling.

Uncontrollability Algorithm 1 can also be easily extended to account for uncontrollable RAS dynamics, where uncontrollability is defined w.r.t. the timing and/or the routing of some process-advancing events; i.e., these events may occur spontaneously as long as the requested resources are available or the further routing of certain processes might be externally dictated by the need for special treatment or rework. This uncontrollable behavior necessitates a redefinition of the notion of "state unsafety": under this new regime, a feasible state s is unsafe as soon as there exists an uncontrollable event that is enabled at s and leads to another unsafe state. Therefore, for all partial transition relations Δ_{σ_u} where σ_u is an uncontrollable event, unsafe states can be directly identified by backtracing from the currently identified unsafe states without the need to perform the other symbolic operations in Lines 14-27 of Algorithm 1. In fact, starting from an unsafe state, this backward search on each Δ_{σ_u} can be carried out repeatedly until a fixed-point is reached; all the states reached by this process are subsequently entered into set $\chi_{U_{new}}$. On the other hand,

[9] To facilitate the review process for this manuscript, we also provide this analysis in [16], which is accessible from the author websites.

[10] The experiments were carried out on a standard desktop, (2.66 GHz Intel Core Quad CPU, 8GB RAM) running Windows 7.

[11] More specifically, the quanity η_t reported in Column 7 is computed by $\eta_t = (t_o - t_n)/t_o$ and similarly, the quantity η_m of Column 8 is computed by $\eta_m = (\zeta_o - \zeta_n)/\zeta_o$.

[12] Even in the cases where the reported relative increase might seem pretty large, it can be checked in the provided data that the actual increase in the computation time, in terms of the actual time values, is not very big.

TABLE I: A set of computational results demonstrating the efficiency of the presented algorithm.

$\|R\|$	$\|FB\|$	t_o	ζ_o	t_n	ζ_n	η_t	η_m
799, 071	283, 962	7s	283, 962	9s	118, 509	-28%	58%
1, 659, 342	800, 940	42s	796, 123	49s	369, 875	-16%	53%
1, 962, 454	761, 399	29s	450, 040	21s	92, 129	27%	79%
3, 436, 211	1, 564, 991	106s	1, 176, 110	84s	520, 275	20%	55%
14, 158, 338	3, 558, 362	152s	1, 561, 971	206s	473, 966	-35%	69%
14, 521, 572	5, 696, 085	642s	4, 999, 572	435s	948, 155	32%	81%
14, 963, 458	5, 989, 367	553s	4, 415, 000	431s	758, 113	22%	82%
22, 212, 582	8, 056, 766	964s	5, 546, 176	436s	1, 099, 411	54%	80%
29, 160, 898	7, 751, 451	237s	2, 685, 162	197s	480, 315	16%	82%
32, 380, 375	14, 320, 225	904s	5, 415, 820	517s	647, 567	42%	88%
1, 712, 672	445, 092	38s	646, 998	29s	77, 865	23%	87%
1, 962, 454	761, 399	25s	649, 984	21s	120, 082	16%	81%
2, 430, 581	741, 764	10s	226, 991	9s	83, 774	10%	63%
2, 939, 463	531, 238	97s	1, 043, 925	79s	144, 508	18%	86%
6, 051, 299	1, 781, 191	32s	575, 720	37s	118, 341	-15%	79%
22, 212, 582	8, 056, 766	815s	5, 182, 290	375s	1, 658, 094	53%	67%
24, 430, 444	6, 000, 747	125s	1, 534, 599	110s	163, 222	12%	89%
29, 160, 898	7, 751, 451	193s	2, 146, 384	240s	450, 029	-24%	79%
106, 509, 798	12, 529, 669	313s	2, 367, 893	358s	382, 689	-14%	83%
596, 212, 152	169, 402, 134	520s	6, 744, 437	515s	431, 616	0.9%	93%
1, 663, 534	262, 514	1s	129, 084	2s	32, 706	-100%	74%
2, 340, 408	603, 701	2s	230, 807	8s	40, 103	-300%	82%
7, 885, 856	594, 828	1s	262, 861	0s	41, 681	100%	84%
30, 397, 584	853, 537	3s	229, 892	3s	28, 648	0%	87%
81, 285, 120	4, 676, 480	0s	120, 387	0s	12, 343	0%	89%
96, 438, 720	6, 321, 838	106s	2, 526, 813	115s	140, 091	-8%	94%
399, 477, 600	122, 636, 544	59s	2, 939, 165	153s	494, 699	-169%	83%
1, 219, 947, 240	72, 055, 380	460s	7, 959, 586	886s	1, 647, 448	-92%	79%
3, 547, 065, 654	93, 980, 859	74s	3, 595, 817	69s	381, 583	6%	89%
3, 749, 923, 584	269, 219, 724	99s	2, 441, 987	145s	112, 904	-46%	95%

events σ_i corresponding to controllable behavior will have their Δ_{σ_i} processed according to the standard logic of Algorithm 1. Due to space limitations, we leave the relevant implementation details to the reader.

V. CONCLUSIONS

This paper has extended the recent results of [4], by presenting a novel symbolic algorithm for computing the boundary unsafe states for RAS instances coming from the class of D/C-RAS. Instead of performing all the symbolic computations on the monolithic transition relation representing the underlying RAS state-space, the proposed algorithm identifies the boundary unsafe states iteratively on a set of less complex partial transition relations, with each such partial relation defined by a particular event in the RAS-modeling EFA. A series of computational experiments has manifested the superiority of this algorithm, especially in terms of memory usage, over the corresponding algorithm in [4]. Furthermore, the algorithm can be (i) easily extended to account for the uncontrollable behavior and (ii) parallelized to take advantage of the contemporary shared-memory multi-core systems.

REFERENCES

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

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

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

[4] Z. Fei, S. Reveliotis, S. Miremadi, and K. Åkesson, "A BDD-based approach for designing maximally permissive deadlock avoidance policies for complex resource allocation systems," *IEEE Trans. on Automation Science and Engineering (to appear)*.

[5] J. R. Burch, E. M. Clarke, and D. E. Long, "Symbolic model checking with partitioned transition relations," in *Proceedings of 1991 Intl. Conf. on VLSI*, 1991.

[6] A. Geldenhuys, Jaco and Valmari, "Techniques for Smaller Intermediary BDDs," in *12th International Conference on Concurrency Theory*, ser. Lecture Notes in Computer Science, M. Larsen, Kim and Nielsen, Ed., vol. 2154. Springer Berlin / Heidelberg, 2001, pp. 233–247.

[7] A. Vahidi, "Efficient analysis of discrete event systems," Ph.D. dissertation, Chalmers University of Technology, Göteborg, Sweden, 2004.

[8] A. Vahidi, B. Lennartson, and M. Fabian, "Efficient Analysis of Large Discrete-Event Systems with Binary Decision Diagrams," in *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 2751–2756.

[9] Z. Fei, S. Miremadi, K. Åkesson, and B. Lennartson, "Efficient Symbolic Supervisor Synthesis for Extended Finite Automata," *IEEE Transactions on Control Systems Technology (accepted for publication)*, 2014.

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

[11] 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.

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

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

[14] A. Nazeem and S. Reveliotis, "Efficient enumeration of minimal unsafe states in complex resource allocation systems," *IEEE Trans. on Automation Science and Engineering*, vol. 11, pp. 111–124, 2014.

[15] 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.

[16] Z. Fei, K. Åkesson, and S. Reveliotis, "A correctness analysis for the algorithm presented in "symbolic computation of boundary unsafe states in complex resource allocation systems using partitioning techniques", by z. fei, k. akesson and s. reveliotis, ieee case 2015 (submitted)," Chalmers University of Technology, Tech. Rep., 2014. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/205470/local_205470.pdf

[17] 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*, Ann Arbor, MI, USA, 2006, pp. 384–385.

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