12th IFAC/IEEE Workshop on Discrete Event Systems
Cachan, France. May 14-16, 2014

A Symbolic Approach for Maximally Permissive Deadlock Avoidance in Complex Resource Allocation Systems

Zhennan Fei * **Spyros Reveliotis** ** **Knut Åkesson** *

* *Chalmers University of Technology, Gothenburg, Sweden (e-mail: {zhennan, knut.akesson}@chalmers.se)*
** *Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: spyros@isye.gatech.edu)*

Abstract: To develop an 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. This paper presents a symbolic approach that provides those critical states. Furthermore, by taking advantage of certain structural properties regarding RAS safety, the presented method avoids the complete exploration of the underlying RAS state-space. Numerical experimentation demonstrates the efficiency of the approach for developing the maximally permissive DAP for complex RAS with large structure and state-spaces, and its potential advantage over similar approaches that employ more conventional representational and computational methods.

Keywords: Resource Allocation System, Discrete Event System, Deadlock Avoidance, Maximal Permissiveness, Supervisory Control Theory, Binary Decision Diagram.

1. INTRODUCTION

In the Discrete Event Systems (DES) literature, the concept of the Resource Allocation System is a well-established abstraction for modeling the resource allocation dynamics that take place in many technological applications (Reveliotis (2005); Zhou and Fanti (2004); Campos et al. (2014)). Following those past developments, in this work, we define a *resource allocation system (RAS)* [1] by a 4-tuple $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ where: (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. 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. (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. 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 a natural framework for its investigation is that of DES Supervisory Control Theory (SCT) (Ramadge and Wonham (1989), Cassandras and Lafortune (2008)). More specifically, in an Finite State

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

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

Automaton (FSA) 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*. 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. Furthermore, 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, a particular approach for the implementation of the maximally permissive deadlock avoidance policy (DAP) for any instantiation of the aforementioned RAS class reduces to the effective enumeration and the efficient storage of the minimal boundary unsafe states of the underlying state space. This approach has been extensively investigated in the recent years, and the major results together with the supporting literature are systematically discussed in Reveliotis and Nazeem (2013).

The work presented in this paper seeks to complement the aforementioned past developments by introducing symbolic methods for the representation of the involved dynamics and of the target sets, and for the execution of the necessary computation. It is well known that, when properly balanced, symbolic representations of a DES state space can effect a dramatic compression of the information that is expressed by this state space compared to its more conventional representations. Furthermore, this compression can also lead to a significant speed-up of the computational algorithms that process this information for various analysis and (control) synthesis purposes. Indeed, the computational results that are presented at the end of this manuscript corroborate these expectations, and demonstrate clearly the effected gains in terms of computational time and memory requirements. On the other hand, due to the imposed page limits, the rest of this document is a rather minimal exposition of the pursued approach and the obtained results. A more expansive and thorough treatment of this material can be found in Fei et al. (2013).

2. PRELIMINARIES

2.1 Extended Finite Automata

The presented work employs the extended finite automaton (EFA) (Sköldstam et al. (2007)) as a formal representation of the RAS dynamics. An EFA 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.

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

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 Fei et al. (2011). For the needs of this manuscript, this construction is briefly illustrated in the following example. 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 Ξ_{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 resources are depicted in Fig.1.

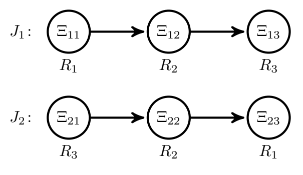

Fig. 1. A simple RAS

In the approach of Fei et al. (2011), each process type is modeled as a distinct EFA. 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. On the other hand, since a process instance that has reached its final stage can always leave the system without any further resource requests, the unloading event is modeled only implicitly through the event that models the process access to its terminal stage(s). 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_{1j}, $j = 1, 2$; 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.