Modeling sequential resource allocation systems using Extended Finite Automata

Modeling Sequential Resource Allocation Systems using Extended Finite Automata

Zhennan Fei, Sajed Miremadi and Knut Åkesson
Department of Signals and Systems, Chalmers University of Technology
SE-412 96 Göteborg, Sweden
{zhennan, miremadi, knut}@chalmers.se

Abstract— **Deadlock avoidance for resource allocation systems (RAS) is a well-established problem in the Discrete Event System (DES) literature. This paper is mainly concerned with modeling the class of Conjunctive / Disjunctive sequential resource allocation systems (C/D RAS) as finite automata extended with variables. The proposed modeling approach allows for modeling multiple instance execution, routing flexibility and failure handling. With an appropriate model of the system, a symbolic approach is then used to synthesize the optimal supervisor, in the least restrictive sense. Furthermore, a set of compact logical formulae can be extracted and attached to the original model, which results in a modular and comprehensible representation of the supervisor.**

I. INTRODUCTION

Deadlock avoidance for (sequential) resource allocation systems (RAS) is a well-established problem in the Discrete Event System (DES) literature. Over the years it has received extensive attention and fruitful approaches have been proposed to develop efficient deadlock avoidance policies. Briefly speaking, this topic is concerned with the coordination of the resource allocation to a set of concurrently executing processes, such that these processes can eventually proceed to accomplishment without the need of external intervention. In practice, this problem lies in many contemporary workflow management systems, such as Automated Guided Vehicle systems, Urban Monorail Transport systems and Internet-based Computing systems.

Conceptually, deadlock avoidance is one of three strategies (prevention, detection and recovery, deadlock avoidance) for dealing with the problem of the RAS deadlock [1]. Though easily implemented, the prevention strategy might stifle the operational flexibility of the system via constraining the RAS structure. With respect to the detection and recovery strategy, the deadlock can occur but the system is equipped with a monitoring mechanism that can trigger an exception handling procedure that resolves it. The deadlock avoidance strategy, making use of the current allocation of the system resources and the available knowledge about the structure of process types, avoids the development of circular waiting patterns from which deadlock is unavoidable. Meanwhile, the strategy maintains the maximum operational flexibility of the considered RAS.

In order to develop deadlock-free control policies for RAS, one way is to utilize the standardized synthesis algorithm provided by Supervisory Control Theory (SCT) [2], [3]. However, it is known that the establishment of such optimal supervisor belongs to the class of NP-complete problems [4], [5]. Hereby researches have been performed to alleviate this problem. To our knowledge, the relevant researches can be classified into two categories. The work in the first category aims at the development of sub-optimal, but computationally efficient deadlock avoidance policies, e.g., [6] and [7]. On the other hand, the work in the other category takes a more aggressive attitude and seeks to synthesize the optimal supervisor directly by using compact and operation-efficient data structures to represent the considered systems. In [8], the authors propose an approach to synthesizing the minimally restrictive non-blocking supervisor by developing a compact representation of the information that is necessary for the characterization of the optimal result. In [9], an approach where the non-blocking, controllable and minimally restrictive supervisor is computed symbolically, is presented.

Nevertheless, from a modeling standpoint, to be able to perform the automatic synthesis with less time and cost, it is important to have an appropriate model of the system. An appropriate model captures what is necessary for solving the particular problem while disregards the irrelevant information. Typically, Deterministic Finite Automata (DFA) or Petri nets are utilized as the modeling formalism. In [10], an approach is presented to use DFA to model the operation-based recipe as the plant. Besides, by specifying alternative branches for a single operation and introducing uncontrollable events, uncontrollable behavior can be modeled. The disadvantage is that the DFA model does not support multiple-instance execution. In [11], a method is presented to avoid deadlocks in a manufacturing system. Each product and resource is modeled by a labeled Petri net. Each job is described by a sequence of operations where each operation will be produced by an identified resource. The advantage is that it is easy to implement and scales well to large systems. The disadvantage is that it is not possible to specify alternative resources or alternative operations.

In the context of the aforementioned research developments, motivated by the above remarks, particularly inspired by [10], we present a new approach to modeling the resource allocation systems. The proposed approach models the considered C/D RAS into a set of Extended Finite Automata, introduced in [12], which are ordinary automata augmented with variables, guard formulas and action functions.

II. PRELIMINARIES

In this section, some preliminaries used throughout the rest of the paper are provided and briefly explained.

A. Conjunctive/Disjunctive Resource Allocation System

Definition II.1: A *Conjunctive / Disjunctive resource allocation system* is formally defined by a 4-tuple [1], [8]:

$$\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$$

where

- $\mathcal{R} = \{R_1, \ldots, R_m\}$ is the set of the system *resource types*;
- $C : \mathcal{R} \to Z^{+}-$ is the *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 i;
- $\mathcal{P} = \{\Pi_1, \ldots, \Pi_n\}$ denotes the set of the system *process types* supported by the considered system configuration. Each process type Π_j is a composite element itself, in particular, $\Pi_j = \langle \mathcal{S}_j, \mathcal{G}_j \rangle$, where $\mathcal{S}_j = \{\Xi_{j1}, \ldots, \Xi_{jl_j}\}$ denotes a set of *processing stages* involved in the definition of process type Π_j; $\mathcal{G}_j$ is an *acyclic diagraph* with the node set equal to $\mathcal{S}_j$. Every path in $\mathcal{G}_j$ connecting a "source" to a "sink" node corresponds to an execution sequence of Π_j;
- $\mathcal{A} : \bigcup_{j=1}^{n} \mathcal{S}_j \to \prod_{i=1}^{m} \{0, \ldots, C_i\}$ is the *resource allocation function* associating every processing stage Ξ_{jk} with the resource allocation vector $\mathcal{A}(\Xi_{jk}) \equiv A_{jk}$ required for its execution.

Furthermore, it is assumed that after a process instance accomplishes a non-terminal stage Ξ_{jk}, it must allocate the entire set of resources implied by the resource allocation request, in order to advance to the next stage. As soon as the requested resources are allocated, it releases all allocated resources that are not needed any more. The considered resource allocation protocol further guarantees that no resource type $R_i \in \mathcal{R}$ is over-allocated with respect to the capacity C_i at any processing stage.

B. Extended Finite Automaton (EFA)

Definition II.2: An extended finite automaton E is a 4-tuple:

$$E = \langle Q, \Sigma, \to, q_0 \rangle$$

where

- $Q : L \times V$ is the extended finite set of states, where L is the set of *locations* and $V = V^1 \times \ldots \times V^n$ is the finite domain of the *variables* $v = (v^1, \ldots, v^n)$;
- Σ is a nonempty finite set of events (the alphabet);
- $\to \subseteq L \times \Sigma \times G \times A \times L$ is the transition relation, where G is a set of guard predicates on $L \times V$ and $A = \{a \mid a$: a function from V to $V\}$ is a collection of action functions;
- $q_0 = (l_0, v_0) \in L \times V$ is the initial state, where l_0 is the initial location while v_0 the initial values of the variables.

For convenience, the symbol ξ is used to denote implicit actions that do not update the value of variables. For instance, if $a^i(v^j) = \xi$, it means that action a^i does not update the variable v^j.

C. Supervisory Control Theory (SCT)

Supervisory Control Theory [2], [3] is a model-based framework, given a system model to be controlled, the plant P and the intended behavior, the specification Sp, a supervisor S can be automatically synthesized, guaranteeing that the behavior of P always fulfills Sp. Here the aforementioned supervisor S is said to be minimally restrictive, meaning the plant is given the greatest amount of freedom to generate events without violating the specification. If the plant is given as a number of sub-plants $P_1, \ldots, P_n$, the plant P can be obtained by performing the *full synchronous composition* [12] operation $\parallel$ on $P_1, \ldots, P_n$. Similarly, $Sp = Sp_1 \parallel \ldots \parallel Sp_m$.

In SCT, events in the alphabet Σ can either be controllable or uncontrollable. Thus the alphabet can be divided into two disjoint subsets, the controllable event set Σ_c, and the uncontrollable event set Σ_u. In addition, there are two properties [2], [3] that the supervisor might or should have:

- *Non-blocking*: This is a progress property enforced by the supervisor S, which guarantees that from any state, there is a path to one of the *marked states*. Referring to an EFA, the marked states are defined as $L^m \times V^m \subseteq L \times V$, where L^m and V^m denote the marked locations and values respectively.
- *Controllability*: Let Σ_u be the set of uncontrollable event set. The supervisor S is never allowed to disable any uncontrollable event in Σ_u that might be generated by the plant P.

Due to the NP-hardness of computing the optimal supervisor for the considered RAS, most of the currently proposed approaches aim at the development of suboptimal, but computationally efficient supervisors. In our work, we aim at the optimal solution. In [9], a framework is presented, where the users model a system by EFAs and obtain the supervisor modularly in form of EFAs. The only difference between the original and final EFAs is that the guards are extended in the latter model. The main advantage of this approach is that the final supervisor becomes more comprehensible for the users. In addition, the users will remain in the same model domain, i.e., EFAs, both when they model the system and when they obtain the supervisor. To be able to handle large systems, all computations are performed symbolically using Binary Decision Diagrams (BDDs) [13]. The procedure is carried out in five main steps shown in Fig. 1.

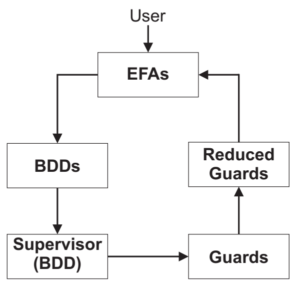


Fig. 1. Process overview of the symbolic supervisory synthesis on EFAs.

Initially, the EFAs are converted to BDDs. Based on the BDDs, the supervisor is computed, which is used to extract the guards. To make the guards more tractable for the users, the guards will then be reduced by some heuristic techniques. Finally, the reduced guards will be attached to the original EFAs. This procedure can be repeated iteratively, making it possible for the users to do further modifications on the obtained supervisor and compute the new supervisor. The detail of this approach is beyond the scope. In this paper, we mainly focus on the modeling issue and discuss how to model the dynamic behavior of the C/D RAS by using EFAs. For a more detailed elaboration of the guard generation procedure refer to [9].

III. THE PROPOSED MODELING APPROACH

To synthesize the non-blocking, controllable and minimally restrictive supervisor for the considered C/D RAS, it is necessary to have a well-defined model, which appropriately captures the dynamic behavior of resource allocation. In this section, the proposed modeling approach, which is the main contribution to this paper, is presented. Taking input a RAS configuration, the approach can automatically generate a set of extended finite automata, each of which models the resource allocation and deallocation of a process type.

For simplicity and understandability, we start with a simple sequential RAS and first model it as a Petri net. For the readers who might be unfamiliar with Petri net, [3] provides a good introduction. From the Petri net model, the corresponding extended finite automata are then derived. After grasping the basic idea, extended finite automata are directly used to model the remaining C/D RAS.

A. Model Single-Unit (SU) RAS

Example III.1: Consider a flexibly automated robotic cell example, borrowed from [1]. As Fig. 2 shows, the RAS is constituted by two process types Π_1 and Π_2, each of which consists of three processing stages performing the linear structure. The system resource set is $\mathcal{R} = \{R_1, R_2, R_3\}$, with the capacity $C_i = 1, i = 1, 2, 3$. Each processing stage $\Xi_{ij}(i = 1, 2; j = 1, 2, 3)$ requests one single unit of one resource type.

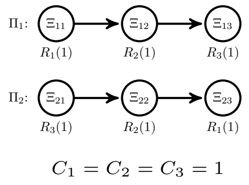


Fig. 2. The considered SU RAS in Section III-A

As the intermediate stage, the considered RAS is first modeled as a Petri net, shown in Fig. 3. For each resource type $R_i, i = 1, 2, 3$, the corresponding resource place is introduced. Initially the number of tokens of each resource place is set equal to its capacity. Similarly, three processing stage places for each process type are introduced to denote the number of process instances executing at the processing stages. For example, the stage places p_{11}, p_{12}, p_{13} map the three processing stages $\Xi_{11}, \Xi_{12}, \Xi_{13}$ of process type Π_1. Moreover, the transitions $t_{11}, t_{12}, \ldots, t_{23}$ depict the resource allocation and deallocation process. The weight of arcs from the resource places to transitions can be considered as the number of requested resource units with respect to various processing stages. From Fig. 3, it can be observed that multiple process instances can be allowed to execute in the Petri net model as long as the resource constraint is satisfied.

With the considered RAS having been modeled as the Petri net, the extended finite automata can be correspondingly derived in the following steps:

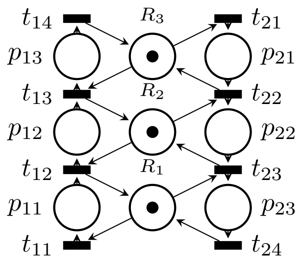


Fig. 3. The Petri net model of the considered RAS in Section III-A

- For each process type $\Pi_i, i = 1, 2$, create an extended finite automaton. To support the multiple instance execution as the Petri net does, each EFA is defined to have only one location and all the transitions labeled with events are added as self-loops. This location is both initial and marked.
- For each resource type (place in the Petri net) $R_i, i = 1, 2, 3$, declare one *resource variable* vR_i denoting the number of available units of R_i. The domain of vR_i is defined to be $\{0, \ldots, C_i\}$, where both of the initial and marked values of vR_i are equal to C_i.
- For each processing stage except the last one of each process type $\Pi_i, i = 1, 2$, declare one *instance variable* $v_{jk}, j = 1, 2$ and $k = 1, 2$, denoting the number of instances executing at the corresponding stage Ξ_{jk}. The domain for each instance variable is defined to be from 0 to the maximal number of executing instances. In this case, since each processing stage only acquires one unit of one resource type, the maximal number of instances at each processing stage is one. Therefore, the domain of all instance variables is defined to be $\{0, 1\}$ where 0 is the initial and marked value.
- Make use of the resource and instance variables defined above to construct the guards and actions. Guards are local formulae which determine whether a process instance can advance to the next processing stage while actions are used to update the available resource units and instances for various processing stages. Finally, the guards and actions are attached to the corresponding transitions of the created EFAs.

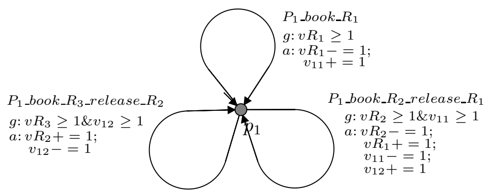


Fig. 4. The EFA model of Π_1 in Section III-A

Fig. 4 and Fig. 5 show the EFAs, which model the process type Π_1 and Π_2 based on the above steps. Here two points must be elaborated. From both the Petri net and the EFA, it can be observed that every time a process instance advances to the non-terminal processing stage, the requested resource allocation and the unused resource deallocation occur simultaneously, which confirms to the assumption made in [1]. The purpose of the supervisor is to prevent the

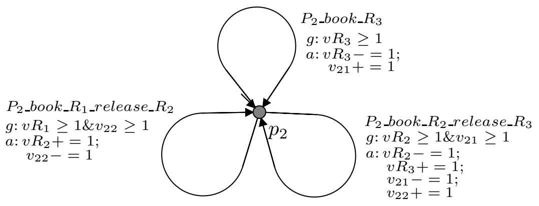


Fig. 5. The EFA model of Π_2 in Section III-A

system from running into blocking situations. Since there is no restriction on these deallocation events as soon as the next requested resources are allocated, we know that all states that have resources waiting to be deallocated cannot be blocking states. Besides, it is noticed that there is no instance variable defined for the terminal processing stage. For the process instance at the terminal processing stage where the requested resources have been allocated, it is assumed that these allocated resources are released immediately. Certainly, this does not model the true behavior of the physical system, but enough information is captured. Reasonably, a model that can be used to find all blocking states need much less information than a model that expresses all possible events and variables, a important reduction of the system size is made.

B. Model C/D RAS

In the previous section, the idea behind the modeling approach is presented through modeling a simple example. Generally speaking, the approach can be summarized by three aspects: EFA creation, variable declaration and guard / action construction. Compared with the Single-Unit (SU) RAS, e.g., the example shown above, modeling is more complicated in the context of the C/D RAS. In this section, after extending the previous example to the one that allows for multiple resource acquisitions and alternative routings, we model such C/D RAS as EFAs.

Example III.2: Fig. 6 shows a C/D RAS which is extended from Example III-A. The considered C/D RAS contains two process types Π_1 and Π_2. Same as before, the processing stages of Π_1 perform the linear structure, but the processing stage Ξ_{12} now allows for alternative resource type acquisition, i.e., either R_2 or R_4. The process type Π_2 is extended to have two alternatives to support the routing flexibility. The processing stage Ξ_{23} requires two types of resources to perform the task. Besides, the capacities of the resource types R_1, R_2, R_3 are increased to 4 and a new resource type R_4 with the capacity 2 is added into the C/D RAS.

In order to model the process type Π_1 by following the previous instructions, the first issue we need to resolve is how to handle the processing stage Ξ_{12}. In particular, how to define the instance variables for it. The processing stage Ξ_{12} allows for the alternative resource type acquisition. The decision that which resource type is allocated to an instance can only be determined dynamically. Besides, when an instance advances to the next stage Ξ_{13}, we cannot know which resource type should be deallocated. Therefore, two instance variables v_{12R_2} and v_{12R_4} need to be declared, which denote the number of instances having acquired R_2

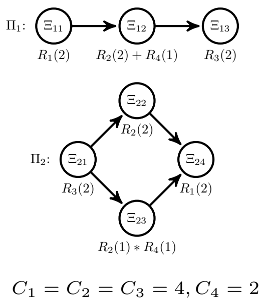


Fig. 6. The considered C/D RAS in Section III-B

and R_4 respectively. Based on the capacities of R_2 and R_4 and the requested units of each type, the domains of these two variables can be obtained, which are $\{0, \ldots, 4\}$ and $\{0, \ldots, 2\}$. With the instance variables defined, the corresponding EFA can be constructed, as Fig. 7 shown.

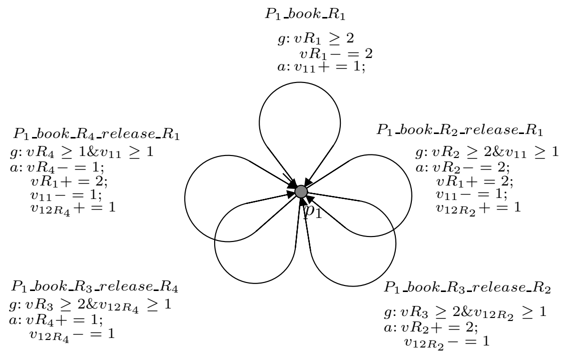
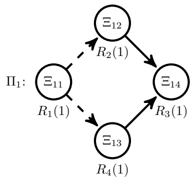

$P_1\_book\_R_1$
$g\colon vR_1 \geq 2$
$vR_1 -= 2$
$a\colon v_{11} += 1;$
$P_1\_book\_R_4\_release\_R_1$
$g\colon vR_4 \geq 1 \& v_{11} \geq 1$
$a\colon vR_4 -= 1;$
$vR_1 += 2;$
$v_{11} -= 1;$
$v_{12R_4} += 1$
$P_1\_book\_R_2\_release\_R_1$
$g\colon vR_2 \geq 2 \& v_{11} \geq 1$
$a\colon vR_2 -= 2;$
$vR_1 += 2;$
$v_{11} -= 1;$
$v_{12R_2} += 1$
p_1
$P_1\_book\_R_3\_release\_R_4$
$g\colon vR_3 \geq 2 \& v_{12R_4} \geq 1$
$a\colon vR_4 += 1;$
$v_{12R_4} -= 1$
$P_1\_book\_R_3\_release\_R_2$
$g\colon vR_3 \geq 2 \& v_{12R_2} \geq 1$
$a\colon vR_2 += 2;$
$v_{12R_2} -= 1$

Fig. 7. The EFA model of Π_1 in Section III-B

With the experience of modeling the behavior of the alternative resource type acquisition, modeling the flexible routings follows the same strategy. Actually, the resultant EFA for Π_2 is similar to the EFA for Π_1, even though the processing stages perform different structures. Note that for the processing stage Ξ_{23}, one variable is enough to model the resource allocation and deallocation for this processing stage. The upper bound of the variable is defined to be the maximal number of instances executing at Ξ_{23} with both resource type units.

C. Model the Abnormal Behavior

The aforementioned modeling methods presume that the considered RAS is totally controllable. Specifically, (1) all the resource allocation events taking place can be disabled by the supervisor if necessary; (2) In a process type presenting routing flexibility, process instances can be conducted by the supervisor to choose different routing options to realize the system flexibility. However, in many contemporary applications, it is necessary to have some form of error handling. When an error occurs for an instance at some processing stage, repair or rework must be performed. Our idea to model such error handling is to introduce alternative branches after the necessary processing stage. Being different from modeling the routing flexibility, the events corresponding to these uncontrollable branches are modeled as uncontrollable events. The supervisor cannot influence which branch to



$$C_1 = C_2 = C_3 = C_4 = 1$$

Fig. 8. The process type Π_1 with error handling

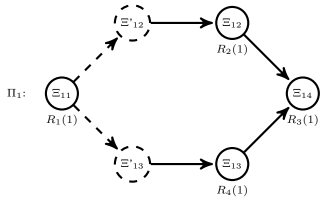


Fig. 9. The process type Π_1 with imaginative stages

choose. Hereby, it must assure that there exists a non-blocking path for all branches.

Example III.3: Consider the process type Π_1 of the example presented in Section III-A. At this time, we suppose that an error may occur at the processing stage Ξ_{11} and needs to be handled by one unit of R_4. To distinguish from the flexible routing options, these two uncontrollable branches are described by dashed lines, as Fig. 8 shows.

As mentioned above, to model such alternatives, two uncontrollable events are introduced. Note that these two uncontrollable events have nothing to do with the resource allocation and deallocation. They are merely used to indicate the success and failure of an instance executing at Ξ_{11}. To assure that failed instances enter Ξ_{13} while successfully executed ones enter Ξ_{12}, two more EFA variables need to be declared. These two variables can be thought of as the variables of two imaginative stages, as Fig. 9 shows. An instance at either of these two imaginative stages still possesses the resources allocated to Ξ_{11}. Once the resource constraint is satisfied, it enters the next stage while the unused resources in Ξ_{11} are deallocated.

Fig. 10 shows the resultant EFA which models the process type Π_1 with error handling. Two uncontrollable events $!normal$ and $!abnormal$ indicate the success and failure of the instances executing in Ξ_{11}. Two variables iv_{12} and iv_{13} denote the number of instances which need enter Ξ_{12} and Ξ_{13} respectively. Note that it is only when an instance enters the next stage Ξ_{12} or Ξ_{13}, the resource (1 unit of R_1) is deallocated.

IV. Case Studies

In this section, the modeling approach discussed in the previous section is applied to several examples. After the considered systems are modeled, it is ready to synthesize the optimal supervisor. To give an overview of how the extracted guards look like, as a first example, we consider the application of this symbolic approach to those two EFAs, shown in Fig. 4 and 5 . As a result, two guards $v_{22} == 0\&v_{23} == 0$ for $P_1\_book\_R_1$ and $v_{11} == 0\&v_{12} == 0$ for $P_2\_book\_R_3$ are extracted. The first guard puts an additional restriction on the process type Π_1 when the resource R_1 is going to be

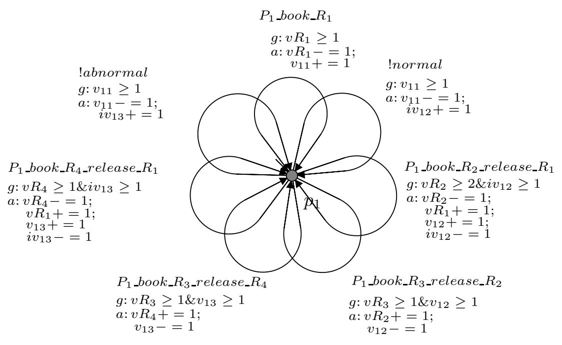


Fig. 10. The EFA modeling error handling for Π_1

allocated. Particularly, R_1 can be allocated only if (1) There is an available unit of R_1 (the original guard); (2) There is no instance of Π_2 executing at processing stages Ξ_{22} and Ξ_{23} (the generated guard). The second guard for Π_2 can be interpreted similarly. These two guards are then attached to the EFA model i.e., the transitions labeled with the events ($P_1\_book\_R_1$ and $P_2\_book\_R_3$). At this moment, the EFA model can be viewed as the supervisor, which prevents the system from reaching the deadlock states.

A. Benchmark Examples

Consider a flexible manufacturing system configuration, shown in Fig. 11, introduced in [14]. The D/C RAS is constituted of three process types and seven resource types $\mathcal{R} = \{R_1, \ldots, R_7\}$ with the corresponding capacities and resource request shown in Fig. 11.

The model of the RAS is constituted by three EFAs. Since there is no processing stage which requests alternative resource types, each non-terminal processing stage needs one instance variable. Besides, seven resource variables are declared, each of which corresponds to one resource type. The construction of guards and actions is similar to the examples discussed in Section III-A and III-B. Then we feed the model into the symbolic approach [9]. As a result, the optimal supervisor contains 8761 states after 3019 states are detected as blocking states and excluded from 11880 reachable states and the computation time is less than 2 seconds[1]. Besides, 10 guards with the average term size 79, are extracted. It should be mentioned that here the result computed from the EFA model is different from that in [8]. Recall that in previous section, the assumption is made that for the terminal processing stages, the resource allocation and deallocation occur simultaneously. Hence, there is no need to declare unnecessary variables for the terminal stages. We consider it as a reduction of the system size. Actually, if we define the instance variables for the terminal stages and add the extra transitions into the model, the number of states for the supervisor would be the same as [8].

Next, we extend the aforementioned flexible manufacturing system configuration example to make it more complicated, shown in Fig. 12. In particular, two more resource types R_8 and R_9 are included and the capacities for other resource types are increased. Among those processing stages,

[1]The experiment was carried out on a standard desktop (Core 2 Quad CPU, 2.66 GHz, 4GB RAM) running Windows 7

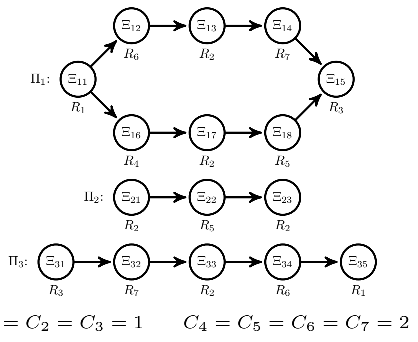
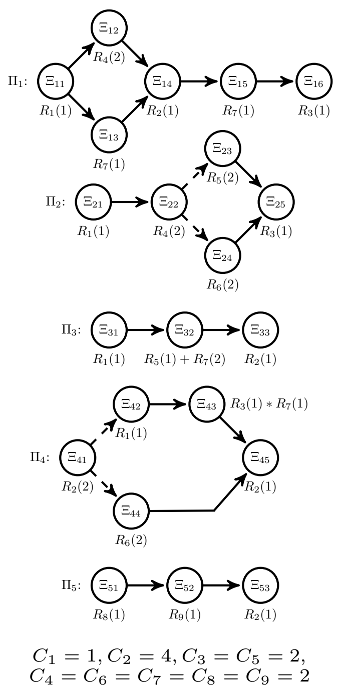


Fig. 11. The flexible manufacturing system configuration in [14]

we suppose that errors might occur at Ξ_{22} or Ξ_{41} and thus the repair needs to be performed.

This extended C/D RAS is managed to be modeled as five EFAs where 31 variables are declared. As a result, the number of reachable states is more than 2.5 million. The supervisor containing 679734 states is obtained within 20 seconds. Around 1.9 million states are identified as either uncontrollable or blocking states. By using the guard generation procedure of the symbolic approach, only 6 guards with the average term size 542 are extracted to prevent the system from reaching 1.9 million problematic states.



Π_1:
Π_2:
Π_3:
Π_4:
Π_5:
$C_1 = 1, C_2 = 4, C_3 = C_5 = 2,$
$C_4 = C_6 = C_7 = C_8 = C_9 = 2$

Fig. 12. The extended D/C-RAS of Fig. 11

V. CONCLUSIONS

In this paper, we present an approach to modeling C/D RAS, one of the RAS classes investigated in the literature. The proposed approach models the C/D RAS as a set of extended finite automata, which allows for multiple instance execution, routing flexibility and error handling. With the model being well-defined, a symbolic supervisory synthesis approach is utilized to compute the optimal supervisor. Furthermore, a set of compact guards is extracted and attached to the model, which results in a modular and comprehensive supervisor.

As future work, the presented approach can be refined and improved from the following two aspects:

- The algorithm needs to be implemented and integrated in the supervisory control tool Supremica [15].
- Investigate alternative compact variable encodings for resource types and processing stages. It is believed that the size of generated guards from the symbolic supervisory synthesis is more or less determined by the number of declared EFA variables. If less number of variables is declared, guards with the smaller size might be obtained.

VI. ACKNOWLEDGEMENT

The authors gratefully acknowledge Spyros Reveliotis and Ahmed Nazeem at Georgia Tech for their valuable comments on earlier drafts of this paper. This work was carried out within the Wingquist Laboratory VINN Excellence Centre, Chalmers University of Technology, Göeborg, Sweden.

REFERENCES

[1] S. A. Reveliotis, *Real-Time Management of Resource Allocation Systems: A Discrete Event Systems Approach (International Series in Operations Research & Management Science)*. Springer, December 2004.

[2] P. J. G. Ramadge and W. M. Wonham, "The Control of Discrete Event Systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

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

[4] E. M. Gold, "Deadlock prediction: Easy and difficult cases," *SIAM Journal on Computing*, vol. 7, no. 3, pp. 320–336, 1978.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*, 1979.

[6] J. Park and S. A. Reveliotis, "Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings," *IEEE Transactions on Automatic Control*, vol. 46, pp. 1572–1583, 2000.

[7] ——, "Liveness-enforcing supervision for resource allocation systems with uncontrollable behavior and forbidden states," *IEEE Transactions on Automatic Control*, vol. 18, pp. 234–240, 2002.

[8] A. Nazeem and S. Reveliotis, "A practical approach to the design of maximally permissive liveness-enforcing supervisors for complex resource allocation systems," in *Automation Science and Engineering (CASE)*, 2010, pp. 451 –458.

[9] S. Miremadi, B. Lennartson, and K. Åkesson, "A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata," *Accepted for IEEE Transactions on Control Systems Technology*, 2011.

[10] K. Åkesson, M. Fabian, and A. Vahidi, "Coordination of batches in flexible production," in *American Control Conference*, vol. 4, 2000, pp. 2735 –2739.

[11] Z. Banaszak and B. Krogh, "Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 6, pp. 724 –734, 1990.

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

[13] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[14] J. Ezpeleta, J. Colom, and J. Martinez, "A petri net based deadlock prevention policy for flexible manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 2, pp. 173 –184, 1995.

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