Structural Control of Large-Scale Flexibly Automated Manufacturing Systems

Spyros A. Reveliotis
School of Industrial & Systems Engineering
Georgia Institute of Technology

Mark A. Lawley
School of Industrial Engineering
Purdue University

Placid M. Ferreira
Department of Mechanical & Industrial Engineering
University of Illinois

Abstract

Current strategic and technological trends in discrete-part manufacturing require extensive and flexible automation of the underlying production systems. However, even though a great deal of work has been done to facilitate manufacturing automation at the hardware component level, currently there is no adequately developed control methodology for these environments. In particular, it has recently been realized by the manufacturing community that any successful attempt towards the extensive automation of these environments requires the establishment of logically correct and robust behavior from the underlying system. The resulting line of research is complementary to the most traditional performance-oriented control, and it has come to be known as the structural control of the contemporary flexibly automated production environment. In this chapter, we initially discuss the role of structural control in the broader context of real-time control of flexible manufacturing systems (FMS), and in continuation, we present the major results of our research program on manufacturing system deadlock, currently the predominant structural control problem in the manufacturing research literature. In the concluding section, we also indicate how these results should be integrated in contemporary FMS controllers, and highlight directions for future research in the FMS structural control area.

1 Introduction

Current trends in Discrete-Part Manufacturing Current business and technology trends in discrete-part manufacturing require the effective deployment of large-scale flexibly automated production systems. To a large extent, this is the result of the shift of discrete-part manufacturing strategies from the previously sought *economies of scale* to what has come to be known as *economies of scope* [16, 12, 1]. In this new environment, emphasis is placed on the ability of the system to provide customized products and services, while meeting tighter time constraints, and with highly competitive prices. Large-scale flexible automation is the enabler of the aforestated economies of scope, since, in principle, this technology allows a variety of parts to be produced through the system (*product flexibility*), with each of them being profitably produced over a wide range of production volumes (*volume flexibility*). The high speed, increased accuracy, repeatability, and reconfigurability which are ascribed to these systems, can lead to high resource utilization, while maintaining high responsiveness to any shift in the production objective.

Furthermore, in certain industries the sheer complexity and sensitivity of the manufacturing operations, and/or the increased volume/weight of the transferred material, pose additional requirements for extensive automation of the production environment. For example, in the semiconductor industry, the need for extremely clean environments, combined with the increased size of processed wafers, has turned the modern semiconductor fab into one of the most technologically sophisticated shop floors.

The inadequacy of existing control methodologies The last fifteen years have seen a lot of success in developing the hardware features – computerized process control and factory networking infrastructure – to support the concept of flexibly automated production systems. However, it is becoming increasingly clear to the manufacturing community, that these systems have not fully realized their advertised advantages, yet, primarily because of a lack for an adequate control methodology. Quoting from a recently published paper [32]: *"Currently most control implementations of flexible manufacturing cells (FMC's) have been developed specially to a particular facility, and no generic format or tools exist for systematic creation and planning for control software. In general, these systems are developed as "turnkey" systems by personnel other than the manufacturing engineers responsible by their operation."* The authors go on to identify some of the problems resulting from this ad-hoc approach to the control of current FMC's as: (i) high implementation times and costs, (ii) lack of portability of the resulting control software, and eventually, (iii) limited operational flexibility of the production system. These findings are corroborated by a recent survey regarding manufacturing flexibility in the printed circuit-board industry, where it was found that: *"production technology turned out to be significantly related to mix and new-product flexibility, but with a pattern opposite to what [the authors] expected given the capabilities of the technology ... This touches a very important point: The fact that automated and programmable equipment in [the authors'] sample tends to be used to run the largest production batches, instead of being used in a more flexible way."* [64]

This lack of a sufficient methodology for the design and operation of effective controllers for large-scale flexibly automated production environments is mainly due to the extreme complexity of the underlying control problem. The most prominent causes of this high complexity can be traced at: (i) the large variety of events taking place in such a production system and the time-scales associated with these events, (ii) the stochastic nature of the system operation, (iii) the nondeterminism inherent to flexible systems, and (iv) the discrete system nature that invalidates the application of most well-known classical analytical techniques. As a result, the development of an analytical methodology providing "globally optimal" solutions to the problem FMS control is currently deemed unlikely. In fact, practitioners typically resort to simulation for the synthesis of FMS controllers in real-life applications.

Simulation approaches in FMS production planning and control The major advantage of simulation as an analysis and evaluation tool is extensive modeling flexibility. However, with existing computing technology, the amount of detail to be included in a simulation model is constrained by: (i) the programming skills of the simulation developer and her thorough understanding of the modeled system, (ii) the capabilities and constructs of the employed simulation platform, and (iii) the time available to set up and run the simulation experiments. The third of these constraints becomes especially prominent in a flexibly automated system, where the system operations undergo frequent reconfigurations.

The limitations attributed to simulation-based approaches are partially addressed by the paradigm of Object-Oriented modeling and software development. The Object-Oriented methodology tries to systematically characterize the behavior of the underlying system components and their interaction, and to develop "standardized" libraries of object classes, from which customized models can be rapidly synthesized in a modular fashion (e.g., cf.[4]). Furthermore, in their effort to systematically extract these standardized behaviors, Object-Oriented techniques resort to more formal analytical models and tools, such as Discrete Event Systems (DES) theory, and Generalized Semi-Markov

Process models, thus, moving simulation-based efforts closer to analytically oriented methodologies [6].

Analytical approaches in FMS production planning and control The most prevalent analytical approach to real-time FMS control attempts to *hierarchically decompose* the problem into a number of more manageable subproblems. These subproblems are classified into number of layers, based on the time-scales associated with the corresponding decision horizons. A typical classification scheme discerns (i) *"strategic-level"* decisions, concerning the initial deployment and subsequent expansion(s) of the production environments, (ii) *"tactical-level"* decisions, which determine the allocation patterns of the system production capacity to various products so that external demands are satisfied, and (iii) *"operational-level"* decisions, which coordinate the shop-floor production activity so that the higher level tactical decisions are observed [62, 21]. Among the three (problem) layers of the hierarchical decomposition framework described above, the ones relevant to this discussion are the tactical and the operational; strategic-level decisions are revised over long operational periods (typically measured in years) and therefore, they fall beyond the scope of real-time system control. Hence, in the following, we briefly discuss the tactical planning and the operational control problem, as defined in the hierarchical decomposition framework.

FMS tactical planning Tactical planning attempts to efficiently track external product demand by exploiting the inherent flexibilities and reconfiguration capabilities of the system. The most prevalent framework for addressing this issue was proposed in [62]. It further decomposes the production planning problem into the following subproblems: (i) *part type selection* for immediate and simultaneous processing, (ii) *machine grouping* into identically tooled machine pools, (iii) *establishing production ratios* according to which the different part types must be produced over time, (iv) *resource (pallet and fixture) allocation* to the different part types so that the determined production ratios are observed, and (v) *loading* of the machine tool magazines so that the resulting processing capacity is consistent with the afore-stated decisions. The detailed formulation of each of these problems and their dependencies is determined by the operational details of the underlying environment. The resulting analytical models are typically Integer Programming (IP) [66] formulations, combined with some analytical or simulation-based performance evaluation tool that supports the "pricing" of the various system configurations examined by the IP solution process. Some representative work along these lines can be found in [9, 63, 61] and the references cited therein.

FMS scheduling In hierarchical decomposition, the scheduling problem consists of *part loading* and *dispatching* so that target production objectives are met. Since this decision-making level addresses the real-time operation of the system, the relevant problem formulations assume a fixed system configuration. Unfortunately, the FMS scheduling problem is one of the hardest problems addressed by the manufacturing community. Many researchers from different methodological areas (e.g., "classical OR", control theory, AI, simulation, etc.) have attempted to address it. An interesting classification and general description of all the different approaches taken to the problem is presented in [56]. The authors conclude that the most viable approach, in view of the problem complexity, is *"to rely on the control paradigm to define the nature and objectives of scheduling problems, serving as a problem framework. Within this framework, combined solution heuristics from machine scheduling, resource-constrained project scheduling and AI search could be used to generate candidate schedules. The performance of candidate schedules could be verified using simulation models, where the simulations could be manipulated interactively by an expert scheduler. Such a synthesis would seemingly wed the rich modeling aspects of control and simulation and with the solution-oriented techniques of heuristic algorithms."* Indeed, such a research program was developed in the years following the publication of these ideas, as it is manifested in the work presented in [11, 13, 34, 37, 44, 58, 35, 36]. All this work recognizes that rather simple distributed scheduling schemes seem to be the most viable approach to the scheduling problem. Furthermore, it resorts to control-theoretic techniques to formally analyze the system behavior, measure its performance, and eventually draw some definitive conclusions about the appropriateness of each of these policies in different operational settings.

FMS structural control The hierarchical decomposition framework [62, 21] addresses issues related to the efficient operation and performance optimization of the controlled production system. However, recent attempts to extensively automate these environments indicate that robust and logically correct system behavior must be established before system performance can be addressed. In an analogy to the computing system environments, what needs to be developed is a *"manufacturing operating system"*, which will facilitate the logically correct, efficient and transparent allocation of the system resources to the various applications (i.e., production requirements). The entire suite of problems related to the logical components of such a control system software has been characterized as the *Structural Control* of flexibly automated production systems [48, 41]. The most typical problems currently identified in this area, are: (i) the resolution the *manufacturing-system deadlock*, i.e., the establishment of undisturbed flow for all parts in the system, and (ii) the design of *protocols*

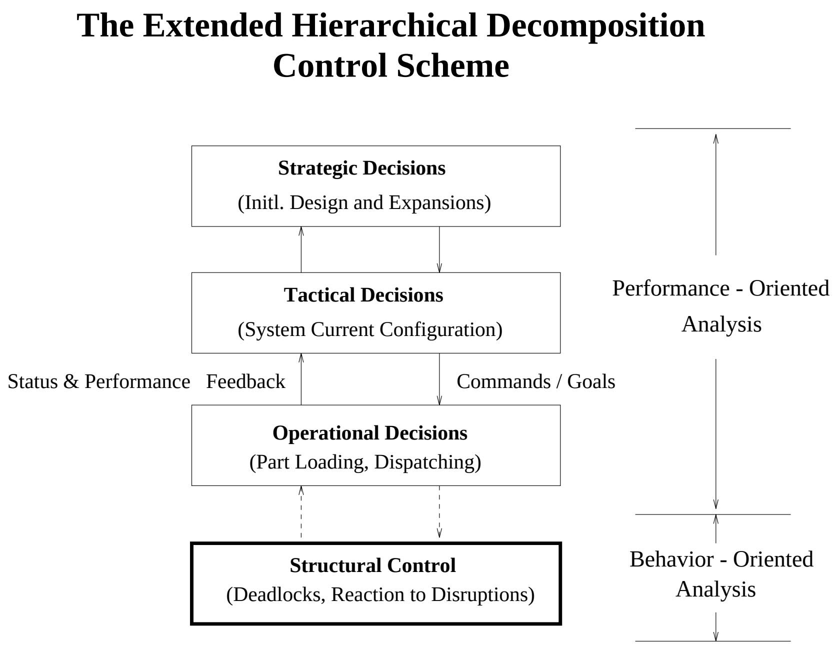

The Extended Hierarchical Decomposition
Control Scheme
Strategic Decisions
(Initl. Design and Expansions)
Tactical Decisions
(System Current Configuration)
Status & Performance Feedback
Commands / Goals
Operational Decisions
(Part Loading, Dispatching)
Structural Control
(Deadlocks, Reaction to Disruptions)
Performance - Oriented Analysis
Behavior - Oriented Analysis

Figure 1: A hierarchical framework for the FMS design and control problem

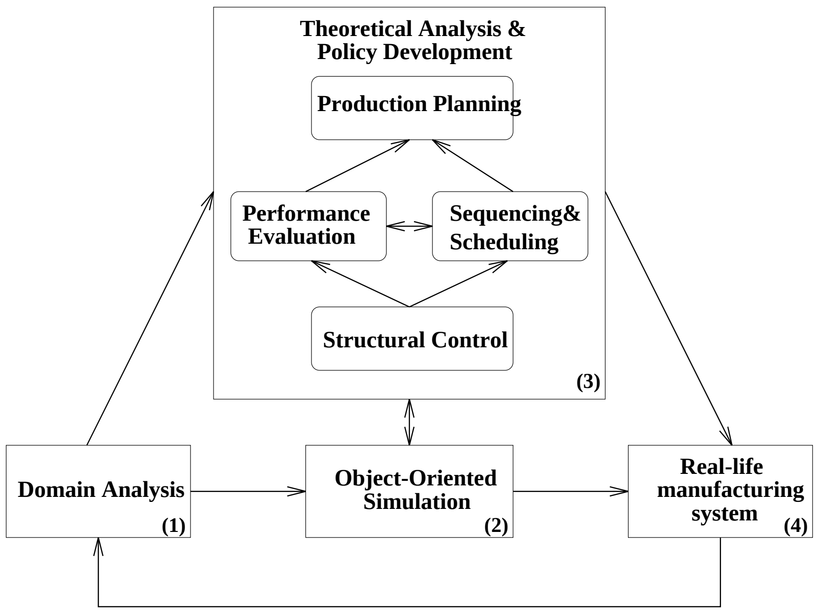


Figure 2: The major (discrete-part) manufacturing system control problems and their interaction

according to which the system should react and recover from different disruptions occurring in it, such as machine breakdowns and the processing of expedited jobs.

Chapter overview Figure 1 presents the hierarchical decomposition control framework, extended to include the new area of structural control. Figure 2 indicates how the structural control problem is interrelated to the rest of manufacturing system control problems, identified in the previous discussion. The rest of this chapter focuses on our recent research results with respect to the manufacturing structural control problem, and in particular, the manufacturing deadlock. Hence, the next section introduces an FMS operational model, demonstrates how deadlock arises in these environments, and overviews the existing literature on the deadlock problem. Section 3 proceeds to the formal modeling and analysis of the problem in the context of Discrete Event Systems (DES) theory [47]. Section 3 will also show that the optimal control of the manufacturing system with respect to deadlock-free operation is, in general, NP-hard [20]. However, in Section 4, we establish that for a large class of manufacturing systems, optimal deadlock avoidance is attainable with polynomial computational cost. Section 5 develops a methodology for designing suboptimal, yet computationally efficient deadlock avoidance policies for the remaining cases that do not admit polynomially optimal solutions. Section 6 considers the efficiency of these suboptimal policies, and suggests a number of ways in which their efficiency can be enhanced. Section 7 concludes with a discussion of additional research topics related to this work. Finally, we note that the discussion in this chapter emphasizes methodological aspects and highlights the key results of the surveyed work. Technical details and formal proofs of the presented results are to be found in the cited publications.

2 The FMS Operational Model and the Manufacturing System Deadlock

2.1 The FMS operational model

The FMS model under consideration is depicted in Figure 3. It consists of a number of *workstations*, $W_1, \ldots, W_N$, each capable of performing some fixed set of e(lementary)-operations, a *Load* and an *Unload* station, and an interconnecting *Material Handling System (MHS)*. Each workstation can be further described by a number of parallel processors/servers (i.e., identically configured machines) and a number of buffering positions holding the parts waiting for or served by the processors. The MHS consists of a number of carriers, each of which can transfer parts between any two workstations;

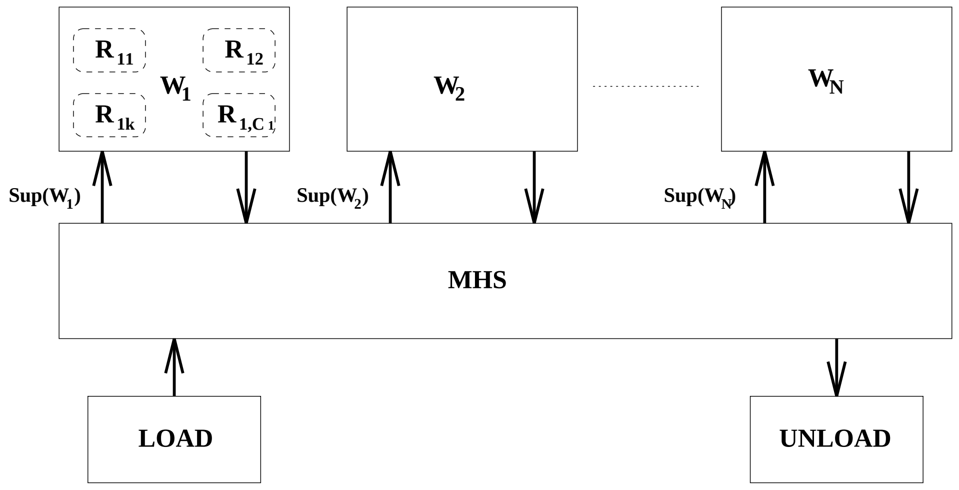
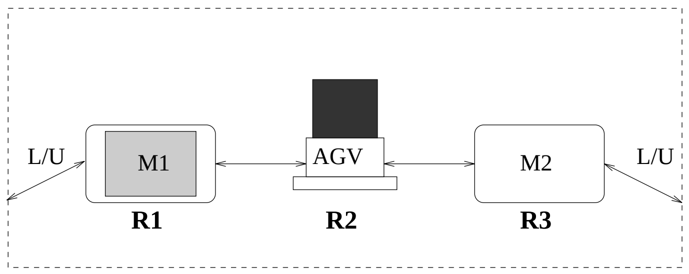


Figure 3: The FMS Operational Model



░ J1 : M1 -> AGV -> M2 ■ J2 : M2 -> AGV -> M1

Figure 4: Example: The FMS Model

an AGV system might be a good example. Jobs are loaded in the FMS through the Load station, and in continuation, they go through a sequence of processing steps performed at the system workstations. Notice that a job can visit a workstation more than once in this sequence. On the other hand, the entire processing performed on a job during a single visit at a workstation is modeled as a single operation. The completed job exits the FMS through the Unload station.

To see how deadlock arises in such an environment, consider the small FMS depicted in Figure 4. It consists of 2 workstations and an AGV, each able to accommodate one workpiece at a time. The jobs supported by the system can take only one of the two routes annotated in the figure, while it is further assumed that all auxiliary equipment required for the processing of these jobs at their different stages, is allocated to them upon their loading into the system. For this small FMS, it is not difficult to see that under the loading pattern depicted in Figure 4, the system is permanently stuck. This is the case, since the job loaded on Workstation M_1 is waiting for the AGV to transport it to the next station M_2, while the AGV is currently occupied by another job which needs to advance to workstation M_1. This permanent blocking that can take place in such manufacturing environments is characterized as the *manufacturing system deadlock*.

Obviously, manufacturing deadlock is a major disruption for the system operation, as its resolution requires external intervention. During its occurrence, the utilization of the resources involved is essentially zero. Furthermore, as it is demonstrated by the above example, this permanent blocking effect results from the fact that every part loaded in the FMS, being a physical entity, has to hold its currently allocated operational space, until it is transferred to a free unit of the resource type it requires next. Specifically, this implicit *hold-and-wait* behavior, can give rise to a *circular-waiting* pattern, where every job involved requires a unit of equipment held by another job in this loop. It is exactly this behavioral pattern that is at the core of the deadlock problem.

Actually, deadlock has also been a problem in the operational environment of the more traditional (non-automated) manufacturing shop floor. However, the manufacturing community has tended to ignore the deadlock problem in that context, since the presence of human operators allowed for an easy solution, namely the preemption of a number of deadlocked jobs to auxiliary buffers so that the remainder could proceed. It is the increased levels of automation and autonomy sought for the contemporary FMS, that require automated solutions to the deadlock problem. Hence, it has been realized by now that establishing nonblocking behavior for the FMS is a prerequisite to any performance optimizing policy in such an environment.

Since the primary cause for the deadlocking effects in contemporary FMS is the *ineffective alloca-*

tion of its buffering capacity, in the subsequent analysis, we shall focus on this capacity of the FMS workstations and MHS, ignoring the detailed operational content of the different processing stages. Hence, all different classes of FMS equipment able to stage a part, will be collectively characterized as the FMS *resources*. However, an additional aspect of the FMS operation to be taken into consideration during the FMS structural analysis, is the management of the system *auxiliary equipment*, i.e., the available pallets, fixtures and cutting tools. Specifically, if there are sufficient units of this kind of equipment, and/or every job is allocated the required auxiliary resources when loaded into the system and it keeps it until exiting, the management of this set of items is not a cause of deadlock. On the other hand, if auxiliary equipment is scarce, and it is allocated to the requesting jobs only prior to executing the corresponding processing step(s) in an exclusive, nonpreemptive manner, then it should be obvious that careless management of these resources can give rise to circular-wait situations, and therefore, to deadlocks. A third feature that is relevant to the characterization of the FMS structural behavior, is whether the routing of a certain part processed through the system is determined on-line (*dynamic* routing) or during the loading of the part into the system (*static* routing). The issue here is that the *machine flexibility* [57] implemented in modern manufacturing processors, as well as, the *operation flexibility* [57] supported by the design of the various products run through the system, allows for the production of a certain part through a number of alternate sequences of processing steps (*process plans* or *routes*). The resulting *routing flexibility* [57] allows for better exploitation of the system production capacity, since the system workload can be better balanced. Hence, from a performance viewpoint, a dynamic routing scheme is more preferable to static one(s). However, dynamic routing introduces an *unpredictable* element to the structural behavior of the different parts running through the system, and makes the development of structural control policies a more difficult problem.

2.2 The underlying resource allocation system (RAS) and the RAS taxonomy

In order to systematically address the problem of FMS deadlock, we must model the FMS operation as a *Resource Allocation System (RAS)*. In general, a resource allocation system consists of a set of concurrently executing processes which, at certain phases of their execution, require the exclusive use of a number of the system resources to successfully run to completion [46]. The resources are in limited supply, and they are characterized as *reusable*, since their allocation and deallocation to requesting processes affect neither their nature nor their quantity. Furthermore, in the FMS case, the resulting RAS can also be characterized as *sequential*, since it is assumed that every process /

job, during its execution, undergoes a *predefined* sequence of resource allocation and deallocation steps. Specifically, every process can be described by a sequence of stages, with every stage defined by the subset of resources required for its successful execution. The detailed structure of the resource requests posed by the executed jobs at their different stages, depends on the FMS operational features discussed in the previous section, i.e., the allocation of the auxiliary equipment, and the employed degree of routing flexibility. In fact, it turns out that the tractability of developing effective structural control policies and the details of the resulting solutions, strongly depend on the way that the FMS is configured with respect to these operational characteristics. This finding has led to the following classification of the FMS-modeling RAS, based on the structure of the allowable resource requests defining these processing stages [40]:

I. Single-Unit (SU) RAS: Every process stage requires only *one* unit from a *single* resource for its successful execution. This model applies to situations where only the *limited buffering capacity* of the FMS equipment is a cause to deadlock.

II. Single-Type (ST) RAS: Every process stage requires an *arbitrary* number of units, but all of the *same* resource type. Similar to the SU-RAS, ST-RAS model FMS in which the only cause of deadlock is the limited buffering capacity of the FMS equipment; however, the ST-RAS allows the modeling of the aggregation of parts running through the system into *tightly connected batches* of varying size.

III. Conjunctive (AND) RAS: Every process stage, to be successfully executed, requires the simultaneous exclusive service of an arbitrary number of units of different resource types. This RAS supports the modeling of the more general case in which *scarce auxiliary equipment* can also be a cause of deadlock.

IV. Disjunctive / Conjunctive (AND/OR) RAS: Every stage is associated with a set of conjunctive requests, the implication being that satisfaction of any of these requests by the system is sufficient for the successful execution of the step. The AND/OR RAS is the model to be used in case that dynamic routing is allowed.

Notice that higher-numbered members of this taxonomy subsume the lower-numbered ones, but the models get more complicated. The results presented in this chapter concern primarily the first (*Single-Unit*) class of this taxonomy. Focusing the discussion on this RAS model allows the exposition of the most significant problem features, without being overwhelmed by the complicating details arising from the increased complexity of higher-level RAS models. The interested reader can find extensions of the results discussed herein to higher level models in [55, 48].

2.3 The nature of the RAS deadlock and generic resolution approaches

The problem of system deadlock was first studied in the context of Computer System Engineering, back in the late '60s and early '70s. The work presented in papers like [27, 26, 10, 28] set the foundation for understanding the problem nature. Specifically, one of the early findings in the study of reusable RAS [10] was that for the development of a deadlock in these systems, the simultaneous satisfaction of the next four conditions is necessary and sufficient:

mutual exclusion i.e., tasks must claim exclusive control of the resources that they require,

"wait for" condition i.e., tasks hold resources already allocated to them while waiting for additional resources,

"no preemption" condition i.e., resources cannot be forcibly removed from the tasks holding them until the resources are used to completion, and

"circular wait" condition i.e., a circular chain of tasks exists, s.t. each task holds one or more resources that are being requested by the next task in the chain.

The identification of the above set of necessary and sufficient conditions for deadlock occurrence, has then driven the efforts for its resolution. In general, all the proposed solutions can be classified into three major categories, namely (i) **Prevention**, (ii) **Detection & Recovery** and (iii) **Avoidance** [46]. *Prevention* methods stipulate that the RAS is *designed* to be deadlock-free by ensuring that the set of necessary conditions for deadlock cannot be simultaneously satisfied at any point in the RAS operation. This requirement is met by imposing a number of restrictions on how a process can request the required resources. *Detection & Recovery* methods allow the RAS to enter a deadlock state. A monitoring mechanism is used to detect a deadlock when it occurs and a resolution algorithm *rolls* the system *back* to a *safe* state by appropriately preempting resources from a number of deadlocked processes. Notice that a RAS state is *safe*, if there exists a sequence of resource acquisition and release operations that allows all the processes in the system to run to completion; detailed definitions of deadlock-related concepts are provided in Section 3. *Avoidance* methods address the problem of deadlocks by controlling how resources are granted to requesting processes. They use *on-line feedback* about the allocation of the system resources, i.e., the RAS state. Given the current

RAS state, a process request is granted only if the resulting state is safe. In principle, such a scheme leads to informed decisions about the safety of a resource allocation operation, allowing the RAS to achieve maximum flexibility.

Among the three methodologies, *avoidance* approaches are considered to be the most appropriate for dealing with the problem of deadlock as it arises in the FMS context [65, 5, 68]. The roll-back required by the detection & recovery approaches when the system is deadlocked is deemed to be too costly in the FMS setting since it implies the temporary "unloading" (preemption) of some currently processed jobs. On the other hand, prevention methods, by being designed off-line, tend to be overly conservative, leading to significant underutilization of the system. The point is that a priori knowledge of the routes of the jobs run through the FMS, makes the information contained in the RAS state theoretically sufficient for the determination of its safety. Therefore, in principle, more efficient FMS deadlock resolution policies can be designed by using on-line feedback w.r.t. the system state and the available information on the anticipated job requirements, during the policy decision making. As we shall see in the next section, however, this statement must be qualified by the findings presented in [25, 3], which indicate that, in the general case, the resolution of the state-safety decision problem – underlying the computation of the maximally permissive FMS DAP – is NP-complete.

In the next section, we report some results with respect to the problem of FMS deadlock avoidance that can be found in the current literature, in addition to the results discussed in the rest of this chapter.

2.4 A literature review of the FMS deadlock problem

In one of the first papers to deal with deadlock in manufacturing systems, [5], the authors remark that avoidance seems to be the most appropriate method for handling deadlocks in an FMS environment, and furthermore, they claim that avoidance methods developed by the computing community are not efficient for FMS applications, since they ignore available information about the process structures and are thus "unduly conservative". A new deadlock avoidance policy, referred to as the *(B-K) Deadlock Avoidance Algorithm (BKDAA)*, is proposed. BKDAA expands on the idea underlying Banker's algorithm[1] [15] – one of the seminal deadlock avoidance policies in computer systems – by exploiting the fact that, in some FMS, certain resources are used exclusively by a single processing step of a single job. These exclusive resources can function as buffers dedicated to the related job

[1]This algorithm will be discussed more extensively in a subsequent section.

instances, thus allowing for the decomposition of the deadlock resolution problem over segments of process routes, and the design of an efficient control algorithm. Using formal Petri Net (PN) modeling [45], the authors show that under BKDAA, the system will never enter a deadlock or a *restricted*[2] deadlock.

PN modeling for the analysis of FMS deadlocks is also used in [65]. This work exploits the concept of PN model *invariants*. The proposed control policy performs a look-ahead search over the set of states reachable from the current state in a certain number of steps and selects as the next system step the one least likely to lead to deadlock. Under this policy, deadlocks are possible, even though the probability of their occurrence is reduced.

In fact, Petri Net related research has produced a series of results that pertain to the FMS deadlock. To deal with the complexity of analyzing Petri Net structures modeling real-life FMS's, researchers in the field have pursued a modular approach. Specifically, the FMS operation is modeled as the interaction of a number of simpler Petri Net modules with well-defined properties. The study of these characteristic properties can lead to some conclusions about the *boundness*, *liveness* and *reversibility* [45] of these modules and the combined networks. Typically, a set of sufficient conditions to be satisfied by the initial state (*marking*) of the system, in order to ensure deadlock-free operation, are eventually identified.[3] The work presented in [70, 24, 31, 19, 17] follows this line of reasoning. In many of these cases, the undertaken analysis has provided significant theoretical insight to the deadlock-related properties and the broader structural behavior of the considered systems – i.e., those that can be modeled and analyzed as a synthesis of the elementary modules. However, in all cases known to us, the sufficiency conditions identified for deadlock-free operation require the enumeration of special structures in the net, like *siphons* and *traps* [45], which is a task of nonpolynomial complexity. Hence, the applicability of these techniques to real-life systems will be severely restricted by computational intractability.

A partial justification for the increased complexity of the Petri Net based approaches discussed above, is that most of them evolved as specializations to the Petri Net framework, of the broader problem of *Supervisory Control of Discrete Event Systems (DES)* [47]. The primary issue undertaken in this paradigm, is the development of a formal framework for the structural analysis and control of DES, based on the theory of *automata and formal languages* [29]. In the Supervisory Control context,

[2]A *restricted* deadlock is defined as a state in which all feasible transitions are inhibited by the constraints imposed by the avoidance policy itself. [5]

[3]Since these approaches resolve deadlock by selecting the initial configuration of the system, they should be classified as *prevention* methods.

the requirement of deadlock-free operation is equivalent to the establishment of *nonblocking* behavior [47] – a prerequisite to any further logical analysis of DES – while the deadlock avoidance problem can be considered as a special case of the *Forbidden State* problem [7]. As it is observed in [7]: "The complexity of the proposed control designs depends mainly on the representation of the forbidden sets. Our methods are very efficient, provided that an efficient, parsimonious representation of forbidden state sets can be found". In fact, this is the more general spirit of the Supervisory Control paradigm, as the emphasis is placed on the *decidability* of problems encountered [29, 59, 60] – i.e., whether a given issue can be effectively / algorithmically resolved – rather than on the provision of computationally efficient algorithms. However, as it will become clear in the ensuing sections of this chapter, the most severe problem in designing practical solutions to the FMS deadlock, is the *lack of a "parsimonious representation"* of the states to be forbidden.

A different approach to the problem is taken in [68, 8]. Here, the authors suggest that a deadlock detection algorithm, presented in [67], be applied to the state that results if a considered request is granted. The request can be granted only if the resulting state is not a deadlock state. This approach is, however, susceptible to restricted deadlocks. Furthermore, the volume of information processed by the deadlock detection algorithm in [67] grows exponentially with the size of the FMS configuration, since the algorithm searches for all possible cycles in a generalized version of the resource allocation graph [28].

The resolution of the FMS deadlock problem is treated in a radically different way in [30]. Recognizing the increased complexity – in fact, NP-completeness – of the exhaustive search for safe allocation sequences, the authors suggest that this search should be restricted along the path that is defined by the applied scheduling (dispatching) policy. Given a resource allocation request, the FMS operation is simulated by first ordering the executed jobs according to a given dispatching rule, and then trying to advance them in that order as much as possible. If the state (*marking*) resulting from this simulation covers the *minimal marking* required to keep the system live [45], then the considered allocation is deemed to be safe, and it is granted to the requesting process.

A thorough analysis of the deadlock problem in Single-Unit RAS where every resource possesses *unit* capacity, is provided in [18]. In that work, the authors use the concept of the *(extended) resource allocation graph* – an analytical structure expressing the interdependencies of the running jobs resulting from the current resource allocation and the job posed requests – to topologically characterize the concepts of deadlock and unsafety in such a system. From this characterization, they proceed to the definition of deadlock avoidance policies as a number of conditions that will guarantee that these problematic structures will never be realized during the system operation. Observance of these conditions during the system operation ensures that it remains deadlock and/or restricted-deadlock-free. Similar results over the entire class of Single-Unit RAS are derived in [69].

Finally, notice that in this survey we have tried to focus on the relatively structured approaches to the problem of FMS deadlock. Given the current urgency of the problem, a number of more ad-hoc solutions, some of them developed specially for certain system / cell configurations, have also appeared in the literature. We have opted not to include them here because of either the specialized or the opportunistic aspect of their nature.

2.5 Developing correct and scalable deadlock avoidance policies for contemporary FMS

With the exception of [5], [30] and some of the policies discussed in [18], the approaches discussed above suffer from problems arising from the NP-complete nature of the state-safety decision problem. Therefore, with these solutions either (i) there is no guarantee that the system will never enter deadlock or restricted deadlock, or (ii) if such a guarantee is provided, it is obtained by (potentially) excessive computational cost (*required computation is exponential in the size of the FMS configuration*). In the first case, we say that the avoidance policy is *not provably correct* and in the second case that it is *not scalable*. It is our position that *deadlock avoidance policies for future technological systems must be, both, provably correct and scalable*. Note that correctness in the context of this discussion implies only that states characterized as safe are indeed safe. There might exist a subset of safe states which will not be recognized as such by a provably correct and scalable avoidance policy. In fact, this is the price one must pay (the compromise one must make) to obtain scalable policies in spite of the NP-complete nature of the underlying decision problem. Provably correct policies that successfully recognize all safe states, are, by definition, *optimal*.

In our research program we have developed a series of FMS deadlock avoidance policies (DAP's) possessing the two properties of correctness and scalability. These results are discussed in Sections 3, 4 and 5. As it has already been pointed out, however, the NP-hardness of the problem of obtaining optimal structural control policies implies that, in the general case, provably correct and scalable policies are going to be *suboptimal*, i.e., preclude some safe states. Therefore, a naturally arising additional requirement is that correctness and scalability must not come at a high cost w.r.t. system performance. Furthermore, the proposed policies must be easily implementable in the context of the current FMS control practices. Efficiency considerations regarding our developed policies is the topic of Section 6.

3 The Single-Unit RAS and the Deadlock Avoidance Problem

3.1 The Single-Unit RAS

The Single-Unit RAS model is defined by a number of *resource types*, denoted by $\mathcal{R} = \{R_i,\ i = 1, \ldots, m\}$, and a number of *process (job) types*, denoted by $\mathcal{J} = \{JT_j,\ j = 1, \ldots, n\}$. Every resource R_i is further characterized by its *capacity* C_i, i.e., a finite positive integer indicating how many units of resource type R_i the RAS possesses. Process type JT_j is defined by a sequence $< JT_{jk},\ k = 1, \ldots, l(j) >$, with element JT_{jk} corresponding to the resource R_q supporting the processing of the k-th step of this type. Thus, JT_{jk} denotes the resource *allocation request* associated with the k-th *stage* of the j-th job type. The sequence of resource allocation requests defining process type JT_j, $j = 1, \ldots, n$, will be referred to as its *route*.[4]

The Single-Unit RAS *state*, $\mathrm{s}(t)$, at a given time instance t, is defined as follows:

Definition 1 *The RAS* state $\mathrm{s}(t)$ *at time t is a vector of dimensionality $D = \sum_{j=1}^{n} l(j)$ – i.e., equal to the total number of distinct route stages in the system –, with component* $\mathrm{s}_i(t)$, $i = 1, \ldots, D$, *being equal to the number of processes executing step k of route JT_q at time t, where q is the largest integer s.t. $i > \sum_{j=1}^{q-1} l(j)$ and $k = i \Leftrightarrow \sum_{j=1}^{q-1} l(j)$.*

Notice that the information contained in the RAS state is sufficient for the determination of the distribution of the resource units to the various processes, as well as the *slack* (or idle) capacity of the system. Furthermore, the finiteness of the resource capacities implies that the set of distinct RAS states is finite; let it be denoted by $\mathcal{S} = \{s^i,\ i = 0, 1, \ldots, Z\}$.

The system state changes in one of the following three ways: (i) *loading* a new process in the system, (ii) *advancing* an already loaded process to its next route stage, and (iii) *unloading* a finished process. During a single state transition only one process can proceed. The resulting step, however, is *feasible* only if the next required resource can be obtained from the system slack capacity. The

[4]As a result, the terms *process type* and *process route* are used interchangeably in this document. To model the effects of *routing flexibility* in a static routing context, a subtler distinction must be made between the concepts of job type and job route: A job type is associated with a distinct product type produced by the system, while a set of job routes characterizes all the possible ways in which this product can be produced through the system. Although this distinction is significant when considering performance aspects, like load balancing of the system, it does not affect the FMS structural behavior, and therefore, it has been downplayed in the proposed FMS modeling.

FMS controller selects the transition to be executed next among the set of feasible transitions. The development of control schemes to guide this selection is the notorious problem of Job-shop *sequencing and scheduling* (cf. Introduction: the scheduling problem). As such, it constitutes the major link between structural and performance-oriented control, and it is beyond the scope of this study. In any case, the selection of a feasible transition by the controller and its execution by the FMS, will be called an *event* in the FMS operation. Furthermore, since the undertaken structural analysis is concerned only with the logical aspects (i.e., the deadlock-free operation) of the system behavior, we focus only on the *sequencing* of these events, ignoring the detailed event-timing. Practically, we assume that the controller can defer making a decision until all transitions that are potentially feasible from the current state have been enabled. Therefore, in the following, we suppress explicit consideration of time.

When time is ignored in the FMS operation, the underlying Single-Unit RAS model can be corresponded to an FSA, where the *event set*, E, consists of all the possible route steps executed in the FMS, the *set of states*, S, corresponds to the set of underlying RAS states, and the *state transition function*, $f \ : \ S \times E \rightarrow S$, is defined by

$$f(s^i, e) = \begin{cases} \text{succ}(s^i, e), & \text{if } e \in \mathcal{F}(s^i) \\ s^i, & \text{o.w.} \end{cases}$$

In the above equation, $\mathcal{F}(s^i)$ denotes the set of feasible events in state s^i, and the function $\text{succ}(s^i, e)$ returns the state that results when event $e \in \mathcal{F}(s^i)$ takes place with the FMS being in state s^i. The *initial* and *final* states of this automaton are identified by state s^0, the state in which the FMS is idle and empty of jobs. Hence, the *language accepted by the FMS* consists of those input (controller command) strings that, starting from the empty state, leave the FMS in the same idle condition. In a physical interpretation, these strings correspond to complete production runs.

Example We elucidate the previously defined concepts by returning to the example of Figure 4. The system resource set is $\mathcal{R} = \{R_1, R_2, R_3\}$, with $C_i = 1, \ i = 1, 2, 3$. The job types supported by the system can be formally described as:

$$JT_1 =< R_1, \ R_2, \ R_3 >$$

$$JT_2 =< R_3, \ R_2, \ R_1 >$$

The system state depicted in Figure 4 is denoted by: $s(t) =< 1\ 0\ 0\ 0\ 1\ 0 >$. Furthermore, it turns out[5] that the size of the entire FMS state space is $Z = 27$, with the state signatures running from 0 to 26. In particular, state $s^0 =< 0\ 0\ 0\ 0\ 0\ 0 >$ denotes the initial empty state. Table 1 enumerates the FMS state transition function and Figure 5 provides the corresponding *State Transition Diagram (STD).* □

3.2 Structural analysis of the Single-Unit RAS

Next, we use the STD of the previous example to analyze the structural/deadlock properties of the Single-Unit RAS model underlying the FMS operation.

State Reachability and Safety A *directed path* in the STD of Figure 5 represents a feasible sequence of events in the FMS. We are mainly interested in paths that start and finish in the empty state s^0. Notice that there is a subset of nodes for which there is no directed path from state s^0; these are shown as dashed nodes in the STD. This implies that when the system is started from empty state, (under normal[6] operation) the states (resource allocation) represented by the dashed nodes will never occur. These states will be referred to as *unreachable*. The remaining states are feasible states under normal operation and will be called *reachable* states. The set of reachable states will be denoted by $\mathcal{S}_r$ and the set of unreachable states will be denoted by $\mathcal{S}_{\bar{r}}$. Notice that $\mathcal{S}_{\bar{r}} = \mathcal{S} \backslash \mathcal{S}_r$. Formally,

Definition 2 *State s^i is* reachable from *state s^j, denoted $s^i \leftarrow s^j$, iff there exists a sequence of events that can bring the system from state s^j to state s^i. In the FSA notation,*

$$\forall s^i, s^j \in \mathcal{S},\ \ s^i \leftarrow s^j \Longleftrightarrow \exists u \in E^* :\ \ f(s^j, u) = s^i$$

Furthermore, a state $s^i \in S$ is a reachable state, *iff $s^i \leftarrow s^0$.*

Another important classification of the STD nodes / states results from the following observation: there are states from which the empty state s^0 is reachable by following a directed path of the STD, and states for which this is not possible. In the STD of Figure 5, the former are lightly shaded while the latter are heavily shaded. If the FMS enters any of the heavily shaded states it will never be able,

[5]It can be shown that, for a SU-RAS, the number of all possible distinct allocations of the system resources is equal to $\prod_{i=1}^{m} \frac{(C_i+Q_i)!}{C_i!Q_i!}$, where Q_i denotes the number of stages supported be resource R_i.

[6]By normal it is meant that the FMS operation observes the assumptions regarding the feasibility of the state transitions, discussed in the previous section.

$i: s^i$	State Vector	Successor States
0	0 0 0 0 0 0	1, 2
1	1 0 0 0 0 0	3, 15
2	0 0 0 1 0 0	4, 15
3	0 1 0 0 0 0	5, 6, 16
4	0 0 0 0 1 0	7, 8, 17
5	1 1 0 0 0 0	9, 18
6	0 0 1 0 0 0	0, 9
7	0 0 0 0 0 1	0, 10
8	0 0 0 1 1 0	10, 19
9	1 0 1 0 0 0	1, 11
10	0 0 0 1 0 1	2, 12
11	0 1 1 0 0 0	3, 13
12	0 0 0 0 1 1	4, 14
13	1 1 1 0 0 0	5
14	0 0 0 1 1 1	8
15	1 0 0 1 0 0	16, 17
16	0 1 0 1 0 0	18
17	1 0 0 0 1 0	19
18	1 1 0 1 0 0	
19	1 0 0 1 1 0	
20	0 0 1 0 0 1	6, 7
21	0 1 0 1 0 1	16
22	1 0 1 0 1 0	17
23	0 1 0 0 0 1	20, 21
24	0 0 1 0 1 0	20, 22
25	0 1 1 0 0 1	11, 23
26	0 0 1 0 1 1	12, 24

Table 1: Example: The FMS State Transition Function

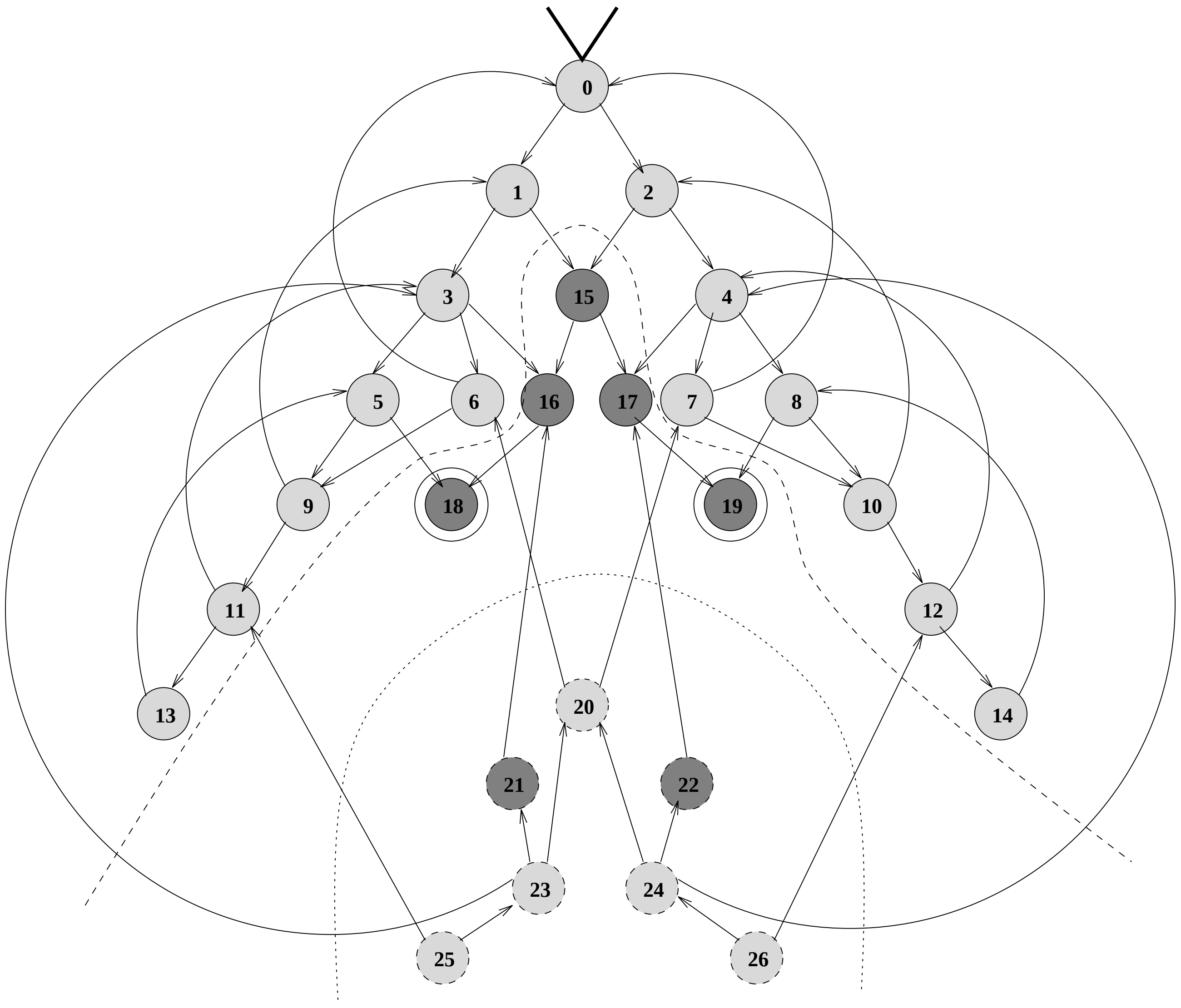


Figure 5: Example: The FMS State Transition Diagram

under normal operation, to complete all running jobs, i.e. become idle and empty. For this reason, the heavily shaded states are characterized as *unsafe*, while the lightly shaded states, which provide accessibility to state s^0, are characterized as *safe*. The set of safe states is denoted by $\mathcal{S}_s \subseteq \mathcal{S}$ and the set of unsafe states is denoted by $\mathcal{S}_{\bar{s}}$. Again, it holds that $\mathcal{S}_{\bar{s}} = \mathcal{S} \backslash \mathcal{S}_s$. Formally,

Definition 3 *State s^i is a* safe *state iff state s^0 is reachable from state s^i. A state which is not safe will be called an* unsafe *state. In the FSA notation,*

$$\forall s^i \in \mathcal{S},\ \mathtt{safe}(s^i) \Longleftrightarrow \exists u \in E^* :\ f(s^i, u) = s^0 \Longleftrightarrow s^0 \leftarrow s^i$$

Furthermore, we extend the characterization of safety to FMS transitions emanating from safe states, by characterizing them as safe if they result in a safe state; mathematically,

$$\forall s^i \in \mathcal{S}_s,\ \forall e \in \mathcal{F}(s^i),\ \mathtt{safe}(e/s^i) \Longleftrightarrow f(s^i, e) \in \mathcal{S}_s$$

Finally, we denote the intersection of any two classes resulting from the previous two classifications by $\mathcal{S}_{xy}$ where $x = r, \bar{r}$, and $y = s, \bar{s}$.

FMS Deadlock It has already been stated in the previous section, that an FMS state is a *deadlock* if there exists a set of jobs s.t. every job in the set is waiting for the release of some resources held by some other jobs in the set. This is defined formally below.

Definition 4 *In a single-unit RAS, state s^i is a* partial deadlock*, if a subset of its resources, $\mathcal{DR}^i$, is filled to capacity, and every process holding a unit of these resources requires transfer to another resource in $\mathcal{DR}^i$ for the execution of its next stage.*

The class of FMS deadlock states for a given FMS configuration will be denoted by $\mathcal{S}_d$. Deadlocks are the natural reason for the existence of the unsafe states in the FMS operation. This is established by the following two propositions, formally proven in [48]:

Proposition 1 *An FMS deadlock is an unsafe state.*

Proposition 2 *In the FMS-STD, every directed path that starts from an unsafe state, and does not involve the loading of a new job in the system, results in a deadlock.*

It should be noticed, however, that there can be unsafe states which are not deadlocks. As an example, consider state s^{15} in the STD of Figure 5. This state, although one step away from deadlock, it does not contain a deadlock itself, since both of its loaded jobs can advance to their next requested resource (i.e., the AGV).

3.3 An algebraic FMS state-safety characterization

The definition of the FMS deadlock, together with the two propositions linking the safety and the deadlock concepts, lead to the following *algebraic* characterization of the state-safety problem:

Consider the FMS state $s^i = \mathrm{s}_0$ in which J_0 jobs, $\{j_1, j_2, \ldots, j_{J_0}\}$, are currently loaded in the system. Obviously, $J_0 \leq \sum_{k=1}^{m} C_k$, the total capacity of the FMS. For uniformity of presentation, we include an extra resource R_{m+1} with $C_{m+1} = \infty$. This resource accommodates all jobs exiting the system. Hence, all job routes are augmented by one step: $JR_{j,l'(j)} = R_{m+1},\ j = 1, \ldots, r$, where $l'(j) = l(j) + 1$. Let $U_{it}, i = 1, \ldots, J_0$, denote the resource unit occupied by job j_i at step t. Then, assuming that state s_0 is safe, the total number of steps required for running all the currently loaded jobs to completion, is $T_t = \sum_{i=1}^{J_0}[l'(\mathtt{jr}(U_{i0})) \Leftrightarrow \mathtt{st}(U_{i0})]$, where $\mathtt{jr}(U_{i0})$ and $\mathtt{st}(U_{i0})$ are the functions returning the job type and the running stage of their argument. Finally, let $\{\delta_{ikt}\}$ denote a set of *binary* variables with

$$\delta_{ikt} = \begin{cases} 1 & \text{if job } j_i \text{ occupies a unit of resource } k \text{ at step } t \\ 0 & \text{o.w.} \end{cases}$$

where $i \in \{1, \ldots, J_0\}$, $k \in \{1, \ldots, m+1\}$ and $t \in \{0, \ldots, T_t\}$.

State s_0 *is safe iff the following system in variables* δ_{ikt} *is feasible:*

$$\delta_{ik0} = \begin{cases} 1 & \text{if } \exists q \in \{1, \ldots, C_k\} :\ U_{i0} = R_{kq} \\ 0 & \text{o.w.} \end{cases} \qquad \forall i, k \tag{1}$$

$$\sum_{i=1}^{J_0} \delta_{ikt} \leq C_k, \quad \forall k, t \tag{2}$$

$$\sum_{k=1}^{m+1} \delta_{ikt} = 1, \quad \forall i, t \tag{3}$$

$$\sum_{t=0}^{T_t} \delta_{i,l'(i),t} \geq 1, \quad \forall i \tag{4}$$

$$\delta_{ikt} \Leftrightarrow (\delta_{ik(t+1)} + \delta_{i,\mathtt{sr}(i,k,p_i),(t+1)}) \leq 0, \quad \forall i, k(\neq m+1), t \tag{5}$$

$$\delta_{ikt} + \delta_{i,\mathtt{sr}(i,k,p_i),(t+1)} + \sum_{j=1}^{J_0} \delta_{j,\mathtt{sr}(i,k,p_i),t} \leq C_{\mathtt{sr}(i,k,p_i)} + 1, \quad \forall i, k(\neq m+1), t \tag{6}$$

$$\delta_{ikt} \in \{0, 1\}, \quad \forall i, k, t \tag{7}$$

In the above equations, $\mathtt{sr}(i, k, p_i)$ is a function returning the resource type required by job j_i for its next processing step, given that it is currently allocated one unit of resource type k for the execution of its p_i-th processing step (s-uccessor r-esource). Equation 1 introduces the description of the initial state s_0 into the program and it represents the set of initial conditions. Equation 2 states that no resource type can hold more jobs than its capacity. Equation 3 expresses the fact that every job instance always requires one complete unit of operational space. Equation 4 simply states that for a RAS state to be safe, it is necessary that every job executes its last step at some point over the considered horizon. Given the convention introduced above, this guarantees that the job has been run to completion and left the system. Equation 5 represents the precedence constraints introduced by the job routes, i.e. every movement of a job in the system must obey its routing sequence. Equation 6 imposes the requirement that a job can proceed to its next step only if there is at least one unit of available capacity from the resource type required during that step. Finally, Equation 7 states the binary nature of the δ variables.

This algebraic characterization of state safety has proven useful in developing a methodology for establishing the correctness of a class of deadlock avoidance policies, discussed in Section 5.

3.4 Deadlock avoidance policies – general definitions

The characterization of state safety and the FMS deadlock by the topological structure of the State Transition Diagram (STD) of the underlying FSA, can also be used to formally define the *avoidance* approach in the resolution of the FMS deadlock. In the following discussion, it is assumed that the FMS always undergoes normal operation, so that only reachable states occur.

The subgraph consisting of the reachable states $\mathcal{S}_r$ and the arcs emanating from them is called the *reachability graph* of the FSA. A DAP must restrict the operation of a given FMS by limiting it to its reachable and safe subspace $\mathcal{S}_{rs}$. Practically, we seek to identify an appropriate set of feasible transitions which when removed from the STD (or equivalently, *disabled* by the DAP), render the unsafe subspace $\mathcal{S}_{r\bar{s}}$ unreachable from state s^0. At the same time, it must be ensured that every state s^i in the remaining graph (i.e., *reachable under the control policy*), is still safe (i.e., there exists a directed path *in the remaining graph* leading from state s^i to s^0). States which are reachable under the DAP and from which progress is inhibited by the policy-imposed constraints and not by the RAS structure, are characterized as *restricted* deadlocks in the deadlock literature [5].

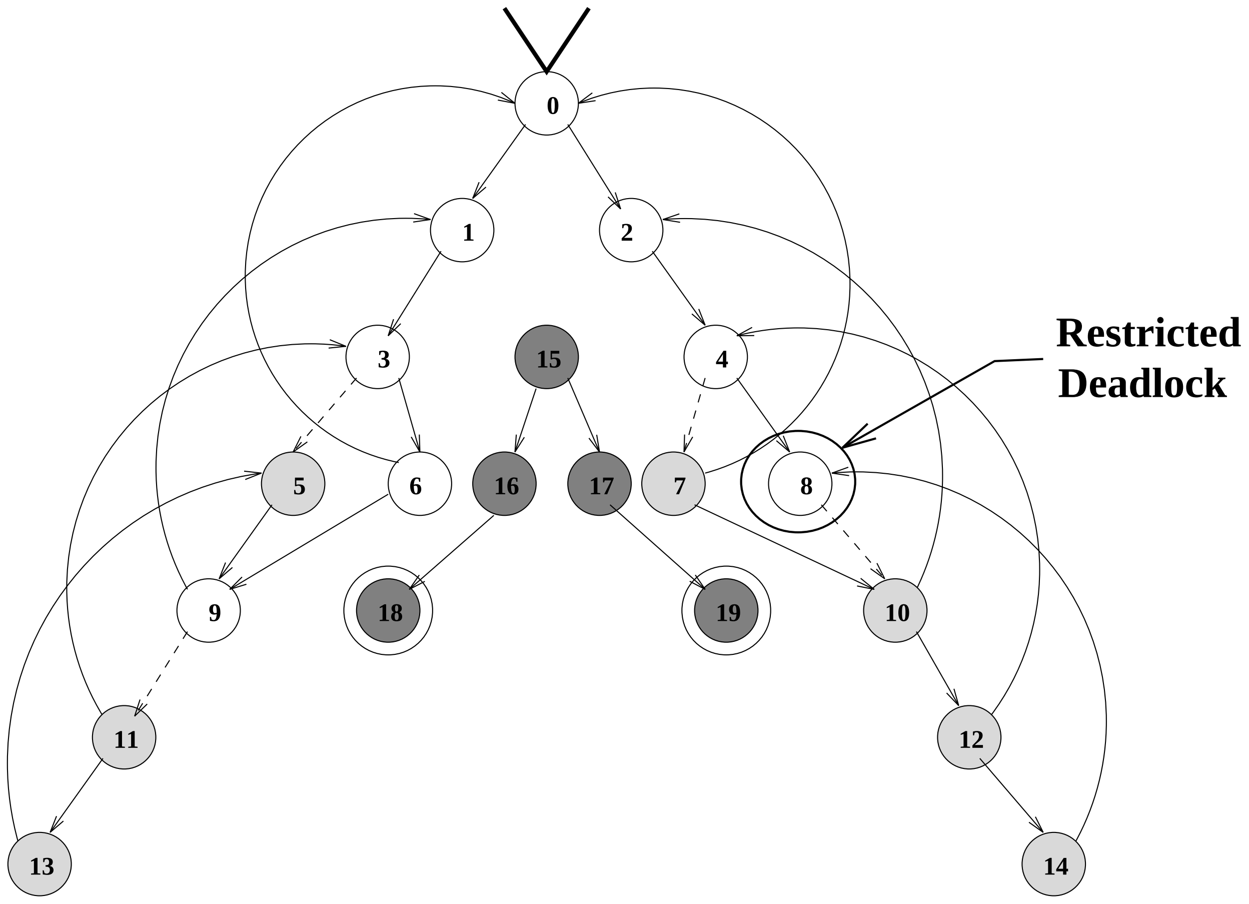


Figure 6: Example: A DAP inducing *restricted* deadlock

An example of a DAP that gives rise to restricted deadlock is presented in Figure 6. This hypothetical policy, defined on the STD of the example FMS of Figure 4, admits (allows access to) only the states corresponding to the white-colored nodes in the depicted STD. Notice that the policy provides accessibility to state s^8, while it disables the only transition out of it, by not admitting state s^{10}. As a result, whenever the system finds itself in state s^8, it is permanently blocked there by the policy logic, itself.

A formal characterization of the above concepts is as follows:

Definition 5 *An* avoidance policy $\mathcal{P}$ *for the FMS is a function*

$$\mathcal{P} : \mathcal{S} \rightarrow 2^E, \quad \mathcal{P}(s^i) = \{e \in \mathcal{F}(s^i) : \ e \text{ is selected by the policy}\}$$

Events $e \in \bigcup_i \mathcal{P}(s^i)$ *are called the* (policy-)enabled *events.*

Definition 6 *Given an avoidance policy* $\mathcal{P}$*, let* $s^i \overset{\mathcal{P}}{\leftarrow} s^j$ *denote the fact that state* s^i *is reachable from state* s^j *through an event sequence which comprises policy-enabled events only. Let* $S_r(\mathcal{P}) = \{s^i : \ s^i \overset{\mathcal{P}}{\leftarrow} s^0\}$ *and* $S_s(\mathcal{P}) = \{s^i : \ s^0 \overset{\mathcal{P}}{\leftarrow} s^i\}$*. Then, policy* $\mathcal{P}$ *is* correct *iff* $S_r(\mathcal{P}) \subseteq S_s(\mathcal{P})$*.*

In words, a DAP is correct iff the policy-reachable subspace $S_r(\mathcal{P})$ is a *strongly connected subgraph containing the idle and empty state* s^0. Notice that the reachable and safe subspace of the uncontrolled system, S_{rs}, possesses this property; in fact, this is the *maximal* subspace possessing this property. This leads us to the concept of the *optimal* DAP. A correct avoidance policy $\mathcal{P}^*$ is *optimal* if the policy restriction on the S_{rs} subspace of the FMS disables *only* those actions that result in unsafe states. Formally,

Definition 7 *The correct avoidance policy* $\mathcal{P}^*$ *is* optimal *iff*

$$\forall s^i \in \mathcal{S}_{rs}, \ \forall e \in \mathcal{F}(s^i), \ \ e \in \mathcal{P}^*(s^i) \Longleftrightarrow f(s^i, e) \in \mathcal{S}_s$$

This characterization of the optimal policy has the following three implications:

i. For a given FMS configuration, the optimal avoidance policy $\mathcal{P}^*$ is *unique*.

ii. $\mathcal{S}_r(\mathcal{P}^*) = \mathcal{S}_{rs}$. Establishing the optimal control policy $\mathcal{P}^*$ is equivalent to removing from the reachability graph those transition arcs that belong to the *cut*[7] $[\mathcal{S}_{rs}, \mathcal{S}_{r\bar{s}}]$. For example, in the STD of Figure 5, the optimal control policy $\mathcal{P}^*$ consists of removing the arcs that emanate from lightly shaded solid nodes and cross the twisted dashed line.

[7]For a definition of this concept refer to [2].

iii. In [25, 3] it is shown that, in the general case, the problem of determining the safety of a RAS state is NP-complete. Since the inclusion of a transition to the optimal avoidance policy $\mathcal{P}^*$ depends on the safety of the successor state, it follows that obtaining policy $\mathcal{P}^*$ is an NP-hard problem [20].

4 Single-Unit RAS admitting Polynomially Computable Optimal DAP

The concluding remark of the last section regarding the complexity of the optimal deadlock avoidance policies in Single-Unit RAS made researchers of the manufacturing system deadlock – including our group – focus their efforts, for a long time, in obtaining good suboptimal policies, i.e., policies that are computationally tractable, and still, relatively efficient with respect to the overall system performance. However, some recently obtained results establish that for a large subclass (in fact, the majority) of Single-Unit RAS, the optimal DAP is polynomially computable, and therefore, implementable in real-time. This section develops the relevant theory.

Specifically, the identification of the aforementioned SU-RAS subclass admitting polynomially computable optimal DAP, resulted from the following two key observations:

1. While the decision problem regarding the safety of a SU-RAS state is, in general, NP-complete, the deadlock detection problem – i.e., whether a given SU-RAS state is deadlock or not – is polynomially computable.

2. There are SU-RAS models in which the set of unsafe states coincides with the set of the deadlock states. In other words, in this class of SU-RAS, there are no unsafe states which do not contain already a deadlock.

Observations 1 and 2 together imply that for the considered subclass of SU-RAS, optimal deadlock avoidance can be obtained by testing for deadlock through one-step lookahead: i.e., the satisfaction of a pending request is simulated, and if the resulting RAS state is deadlock-free, it is concluded that granting the request is a safe step. In fact, this result provides theoretical justification for a claim made by a number of researchers in the field (e.g., [65, 68]), that in the majority of the cases, deadlock avoidance policies based on a finite horizon of look-ahead steps, even though not provably correct, are well behaved.

```
algorithm DDA
begin
    /* Initialize */
    CANDIDATES := RDV;
    STUCK := FALSE;

    /* processing step */
    while (not-empty(CANDIDATES) and not(STUCK)) do
    begin
        STUCK := TRUE;
        for (i:=1 to cardinality(CANDIDATES)) do
        begin
            if (deleteable($R_i$))
            begin
                eliminate($R_i$);
                STUCK := FALSE;
            end
        endfor
    endwhile

    if (not-empty(CANDIDATES))
        state s is a deadlock with resources in CANDIDATES being the deadlocked subset;
    else
        state s is deadlock-free;
end
```

Figure 7: A polynomial deadlock detection algorithm for Single-Unit RAS

4.1 Deadlock detection in Single-Unit RAS

To show the polynomiality of the deadlock detection problem in SU-RAS, it is sufficient to provide an algorithm resolving the problem in polynomial time. Such an algorithm is depicted in Figure 7, and was initially developed in [55]. The algorithm logic is based on the concept of the *Resource Dependency Graph (RDG)*. This is a graph with each vertex V_i, $i = 1, \ldots, m$, corresponding to a system resource type R_i $i = 1, \ldots, m$, and with each edge E_{ij} implying that one of the units of R_i is currently allocated to a process j_k requiring a unit of R_j for its next processing stage. Hence, a partial deadlock in a SU-RAS is depicted by a *closed* subgraph of the RDG in which every node V_i has a number of emanating arcs equal to the capacity of the corresponding resource R_i; such a subgraph is typically called a *knot* in the deadlock literature. The algorithm stores the information contained in the RDG of a given state in a data structure, known as the *Resource Dependency Vector (RDV)*. RDV has one component for every RAS resource type R_i, containing: (i) the number of R_i units allocated to processes in state s, and (ii) a list of the resource types required by the processes currently allocated to R_i, for their next step. Notice that this information can be easily obtained from the state vector and the process routes. Then, the algorithm considers the RDV and tries to identify a subset of resources entangled in a knot. Initially, all system resources are possible candidates. Subsequently, the algorithm goes through a number of scannings of the RDV, and at every scanning it eliminates a number of resources from further consideration. The resources eliminated are those which either (i) are not filled to capacity or (ii) have units allocated to processes which are ready to leave the system or to move to an already eliminated resource. Proceeding in this way, the algorithm either eliminates all resources, in which case the considered state s is deadlock-free, or at a certain scanning no resource is eliminated, in which case state s is a deadlock.

Example To provide a more concrete example of the algorithm logic, consider the SU-RAS state depicted in Figure 8. During the first iteration of the algorithm, resources R_1 and R_4 are eliminated from the CANDIDATES list, since each contains a job ready to leave the system, and therefore, they cannot be entangled in a deadlock. After the second iteration, resource R_3 is also eliminated since it contains jobs waiting upon resource R_4, which was shown to be able to obtain free capacity during the first run. However, during the third iteration, no other resources are eliminated, and therefore, it is concluded that the subset of resources R_2, R_5, R_6 and R_7 are permanently blocked due to a partial deadlock. □

The following proposition, proven in [55], establishes the polynomial complexity of the DDA algorithm:

Proposition 3 *The complexity of the DDA algorithm is $O(m^2 \cdot \bar{C})$, where m is the number of the RAS resource types, and $\bar{C} = \max_{i \in \{1,\ldots,m\}} C_i$ (i.e., the maximum resource capacity).*

4.2 Single-Unit RAS models with no deadlock-free unsafe states

The result regarding the existence of SU-RAS models for which the class of unsafe states coincides with the class of deadlock states is stated in the following theorem:

Theorem 1 *Let a Single-Unit RAS be defined over a resource set R, and a set of job types JT, such that each resource $R_i \in R$ satisfies at least one of the following conditions:*

1. *$C_i > 1$.*

2. *If JT_{jk}, JT_{lm} are two distinct job stages supported by resource R_i, then, (a) either the preceding stages $JT_{j,k-1}$ and $JT_{l,m-1}$ or the succeeding stages $JT_{j,k+1}$ and $JT_{l,m+1}$ are supported by the same resource, or (b) JT_{jk} (JT_{lm}) is the initial/final stage in JT_j (JT_l).*

3. *R_i supports a single job stage JT_{jk}.*

Then, $\mathcal{S}_{ru} \cap (\mathcal{S} \backslash \mathcal{S}_{rd}) = \emptyset$, i.e., every reachable unsafe state is a deadlock state.

A formal proof for this theorem is provided in [33], and it essentially combines similar results initially developed in [55] and [18]. The significance of this result is demonstrated by means of the flexible manufacturing cell depicted in Figure 9. This cell presents the typical layout of many contemporary automated manufacturing cells, consisting of a number of workstations, served by a central robotic manipulator. Each workstation possesses an input and an output buffer, while the manipulator can carry only one part at a time. For such a system, it is clear that deadlock-free operation can be established by controlling the allocation of the buffering capacity of the system workstations, and treating the manipulator as the enabler of the authorized job transfers. Furthermore, the SU-(sub-)RAS defined by the system workstations meets the specifications (1) and/or (2) prescribed in the conditions of Theorem 1, and therefore, in such a system, the optimal deadlock avoidance can be attained through one-step lookahead on the allocation of the buffering capacity of the system workstations.

Taking this whole discussion to a more practical level, we would like to remind the reader of the fact that SU-RAS models apply to FMS environments in which the only cause of deadlocks

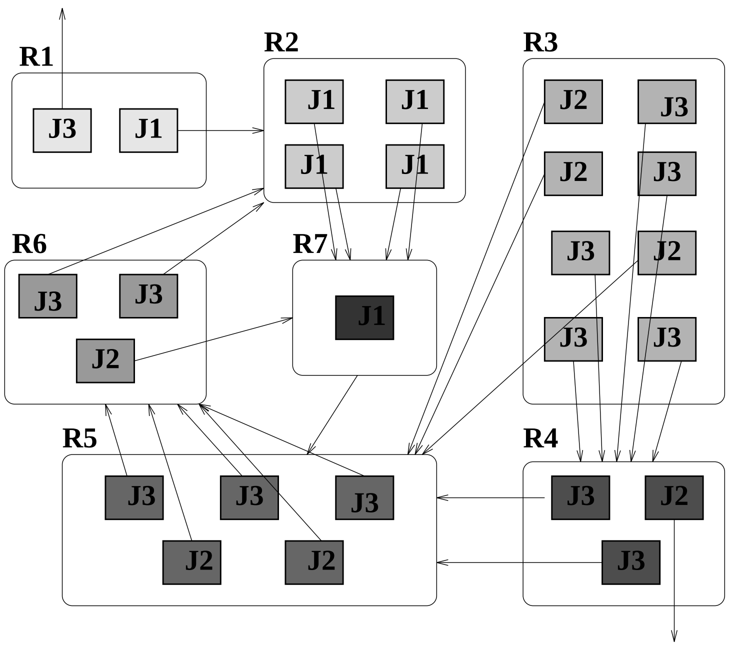


Figure 8: Example: Using the Resource Dependency Graph to detect deadlock

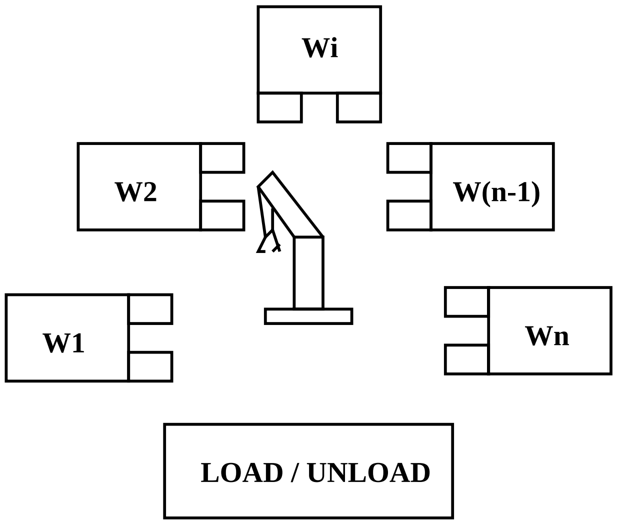


Figure 9: The Flexible Manufacturing Cell layout

is the limited buffering capacity of the FMS equipment (cf. Section 2: the RAS taxonomy). In the light of this remark, the practical implication of the combined results of Proposition 3 and Theorem 1, is that in these FMS environments, *deadlock can be resolved optimally by ensuring that every FMS resource can accommodate at least two parts at a time*. This can be easily achieved with the provision of an input, output or auxiliary buffer. Hence, optimal deadlock-free buffer space allocation in contemporary FMS turns out to be a rather easy problem, under appropriate design of these environments. For the sake of completeness, in the rest of this chapter, we discuss computationally efficient solutions to the problem of deadlock avoidance in SU-RAS, for the cases that the conditions of Theorem 1 are not satisfied. In addition to covering exhaustively the problem of deadlock avoidance over the entire class of SU-RAS, these results have provided useful insight for synthesizing deadlock avoidance policies for more complicated classes of the RAS taxonomy, for which optimality results, similar to those of Theorem 1, are not available. The interested reader is referred to [55, 48].

5 Polynomial-Kernel Deadlock Avoidance Policies for Single-Unit RAS

To deal with the more computationally ill-behaved cases, we adopted an approach based on the concept of *Polynomial-Kernel* policies. Simply stated, this approach requires that, since the target set $\mathcal{S}_s$ is not polynomially recognizable, the system operation should be confined to a subset of these states which is polynomially computable. This state subset is perceived as an easily identifiable (polynomially computable) *kernel* among the set of reachable and safe states, and gives the methodology its name. From an implementational viewpoint, this idea requires the identification of a property $\mathcal{H}(s^i)$, $s^i \in \mathcal{S}$, such that: (i) $\mathcal{H}(s^i) \Rightarrow \texttt{safe}(s^i), \forall s^i \in \mathcal{S}_r$, and (ii) $\mathcal{H}()$ is *polynomially* testable on the system states. Then, by allowing only transitions to states satisfying $\mathcal{H}$, through one-step look-ahead, it can be ensured that the visited states will be safe.

An additional requirement is that the resulting DAP is *correct*, i.e., the policy-reachable subspace must be strongly connected (cf. Section 3: Definition 6). However, this characterization of policy correctness is based on a global view of the system operation, and given the typically large size of the system state space, it is not easily verifiable. A more operable criterion for testing the correctness of Polynomial-Kernel policies is provided by the following theorem:

Theorem 2 *A Polynomial-Kernel DAP is* correct *iff for every state admitted by the policy there* *exists a policy-admissible transition, which, however, does not correspond to the loading of a new job into the system.*

The validity of this theorem primarily results from the observation that at any point in time, the system workload – in terms of processing steps – is finite, and every transition described in the theorem reduces this workload by one unit. Hence, eventually the total workload will be driven to zero, which implies that the system has returned to its home state s^0. A more formal statement and proof of this theorem, by means of the algebraic characterization of state safety provided by Equations 1 – 7 (cf. Section 3), can be found in [52]. Notice, that establishing the policy correctness by means of Theorem 2 resolves concurrently the validity of condition $\mathcal{H}$ as a Polynomial-Kernel identifier for state safety, and the restricted deadlock-free operation of the controlled system.

In the rest of this section, we present three Polynomial-Kernel policies developed in our research program. For each of these policies, we present: (i) the motivation behind the policy defining condition, (ii) the policy defining logic, and (iii) an elucidating example. Formal proofs of the policy correctness and a complexity analysis can be found in the provided references. The efficiency of these policies with respect to the system operational flexibility is discussed in Section 6.

5.1 The Resource Upstream Neighborhood (RUN) policy

It should be obvious that no deadlock would occur in a RAS, if every job were allocated all the resources required for its entire processing upon its loading into the system. In fact, this is a very general *prevention* scheme for deadlock resolution, and as such, it fails to take into consideration any existing information about the RAS structure, being, thus, overly conservative and underutilizing the system resources. The second remark that RUN builds upon, is that if at any point during its sojourn through the system, a job is allocated to a resource of very high – theoretically *infinite* – capacity, then, for the purposes of structural analysis, its route can be decomposed to a number of segments, each of which is defined by two successive visits to the infinitely capacitated resource(s).

Since, in any practical context, resources have finite capacity, RUN exploits the existing job routing information, to implement a *nested, partial* resource reservation system, on the principle that, if there are some resources with higher capacity than others in the system, then they can function as temporary buffers for the jobs that they support. A pictorial representation of RUN reservation scheme is provided in Figure 10. For the detailed formal statement of the policy, we introduce the concept of the *resource upstream neighborhood*:

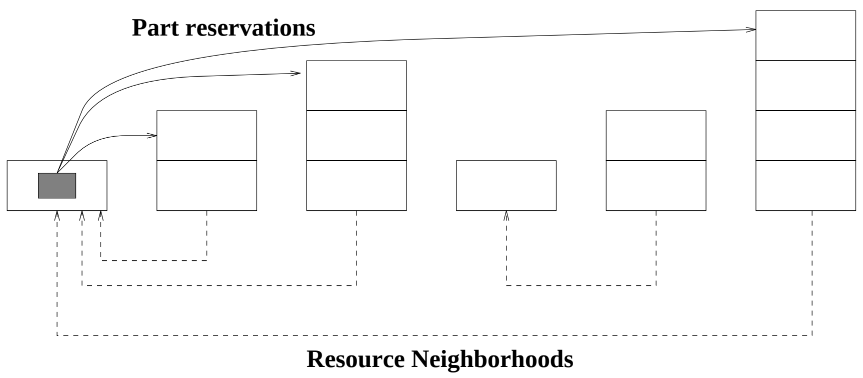


Figure 10: RUN motivation: The partial resource reservation scheme

Definition 8 *[52] The upstream neighborhood of resource R_i consists of all route stages JT_{jk} which are supported by resource R_i, plus all the route stages belonging to the maximal route subsequences immediately preceding each of the aforementioned JT_{jk}, and involving stages JT_{jp} with $C_{R(JT_{jp})} \leq C_i$. A job instance j_j is in the neighborhood of resource R_i iff its current processing stage is in the neighborhood of R_i.*

Then, a formal definition of RUN is as follows:

Definition 9 RUN *[52] A resource allocation state* s *is accepted by RUN DAP iff the number of jobs in the upstream neighborhood of each resource, R_i, does not exceed its buffering capacity, C_i.*

Example We highlight the policy-defining logic and the resource neighborhood construction through an example. Consider the small SU-RAS depicted in Figure 11. This system consists of four resources R_1, R_2, R_3, R_4, with a corresponding capacity vector $\mathbf{C} =< 2, 1, 1, 1 >$. In its current configuration, the system supports the production of three distinct job types, with job routes: $JT_1: R_1 \rightarrow R_2 \rightarrow R_3$, $JT_2: R_3 \rightarrow R_4 \rightarrow R_3$, $JT_3: R_1 \rightarrow R_2$. By applying the logic of Definition 8 to this system, we obtain the neighborhood inclusions indicated by the following incidence matrix:

$$A_{RUN} = \left[\begin{array}{cccccccc} 1 & & & & & & 1 & \\ & 1 & & & & & & 1 \\ & 1 & 1 & 1 & 1 & 1 & & \\ & & & 1 & 1 & & & \end{array} \right] \qquad (8)$$

Each row of the above array corresponds to a resource neighborhood, $N(R_1), \ldots, N(R_4)$. Each column corresponds to a route stage, starting with the stages of job type JT_1, and concatenating the stages of JT_2 and JT_3. Using the definition of the system *state* provided in Section 3 – i.e., a vector

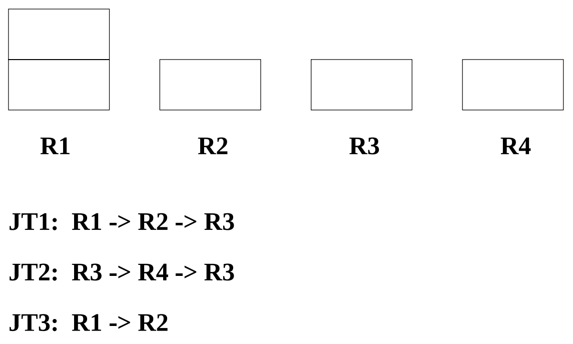


Figure 11: Example: A Single-Unit RAS for RUN and RO DAP demonstration

$\mathrm{s}(t)$ with components corresponding to the distinct job stages JT_{jk}, $j = 1, \ldots, 3$, $k = 1, \ldots, l(j)$, and with $\mathrm{s}_{jk}(t)$ being equal to the number of job instances executing stage JT_{jk} at time step t – it is easy to see that the policy constraints can be expressed by a system of linear inequalities on the system state:

$$A_{RUN}\mathrm{s} \leq \mathbf{C} \tag{9}$$

□

The policy correctness is established in [52]. In fact, in [52] it is also shown that if instead of the partial ordering imposed by resource capacities, any other (partial) ordering of the resource set is used in the neighborhood definition, the resulting policy is still correct. Therefore, RUN logic defines an entire *family* of policies for a given FMS configuration, with each member resulting from a distinct (partial) ordering of the system resources. The exploitation of this result for improving the policy efficiency is discussed in the Section 6. Finally, the policy is *scalable* (i.e., of polynomial complexity in the FMS size, as defined by the number of resources and the distinct route stages of the underlying RAS), since: (i) Construction of the neighborhood sets for a single job type is of complexity no higher than $O(L^2)$, where L is the length of the longest route supported by the system. Therefore, computing the complete resource upstream neighborhoods is of complexity not higher than $O(nL^2)$, where n is the number of job types supported by the system. (ii) Evaluating the admissibility of a RAS state by the policy, requires the verification of m linear inequalities in the D system state variables, where m is the number of system resources, and D is the total number of distinct job stages supported by the system.

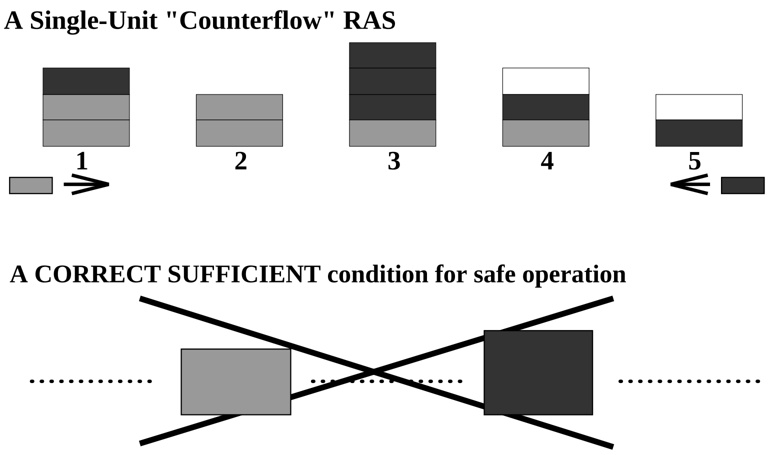

A Single-Unit "Counterflow" RAS
1
2
3
4
5
A CORRECT SUFFICIENT condition for safe operation

Figure 12: RO motivation: A correct sufficient condition for deadlock-free operation of Single-Unit "Counterflow" RAS

5.2 The Resource Ordering (RO) policy

To understand the logic behind the Resource Ordering (RO) policy, let us first concentrate on a subclass of SU-RAS, with the special property that the RAS resources can be numbered so that all job routes correspond to strictly increasing or strictly decreasing resource sequences. We shall characterize this particular class of SU-RAS, as the class of *"counterflow"* systems [40]. It should be easy to see that a sufficient condition for physical and restricted deadlock-free operation in "counterflow" systems, is that no pair of resources (R_i, R_j), with $i < j$, are filled to capacity with the jobs in resource R_i corresponding to ascending resource sequences, and the jobs in resource R_j corresponding to descending resource sequences. These remarks are proven in [40], and are visualized in Figure 12.

Of course, the "counterflow" property is a very restrictive requirement, and the policy would not be of any practical use if it was applicable only to this class of system. It turns out, however, that the policy-motivating idea outlined above can be extended to the more general class of SU-RAS; the solution is to "double-count" job instances for which the remaining route segment is nonmonotonic w.r.t. the resource numbering.

The complete policy definition is as follows:

Definition 10 RO *[43]*

1. *Impose a total ordering on the set of system resources $\mathcal{R}$, i.e., a mapping*

$$o \ : \ \mathcal{R} \rightarrow \{1, \ldots, |\mathcal{R}|\} \ s.t. \ R_i < R_j \Leftrightarrow o(R_i) < o(R_j)$$

We say that "R_i (R_j) is to the left (right) of R_j (R_i)*", iff* $R_i < R_j$.

Furthermore, job stage JT_{jk} is characterized as right (left)*-directed if* $R(JT_{j(x-1)}) < (>) R(JT_{jx})$, $\forall x > k$, *where $R(JT_{jk})$ denotes the resource supporting stage JT_{jk}. A stage which is neither right nor left-directed is an* undirected *stage.*

A job instance is characterized as right, left, or undirected on the basis of its running processing stage.

2. *Let*

   - $RC_i(t) = \{$*right-directed + undirected job instances in* R_i *at time step* $t\}$
   - $LC_i(t) = \{$*left-directed + undirected job instances in* R_i *at time step* $t\}$

3. *A resource allocation state* $\mathrm{s}(t)$ *is accepted by RO iff*

$$\forall i, j \; : \; R_i < R_j \Rightarrow RC_i(t) + LC_j(t) \leq C_i + C_j \Leftrightarrow 1 \tag{10}$$

Example We elucidate the definition of RO DAP, by applying it on the small system of Figure 11. The ordering used in the policy implementation is the *natural* ordering of the system resources, i.e., $o(R_i) = i$, $\forall i$. Furthermore, we observe that job instances executing the last stage of their route can never deadlock the system, since their unloading from the system is always a feasible step. Hence, they can be ignored during the evaluation of the admissibility of a resource allocation state, and therefore, they are omitted during the definition of the content of $RC_i(t)$ and $LC_i(t)$.[8]

It is easy to see that job stages JT_{11}, JT_{12} as well as JT_{31} are right-directed, while job stage JT_{21} is undirected, and job stage JT_{22} is left-directed. Hence, the contents of the RC_i and LC_i counts are defined by the following table:

Resource	RC_i	LC_i
R_1	JT_{11}, JT_{31}	
R_2	JT_{12}	
R_3	JT_{21}	JT_{21}
R_4		JT_{22}

[8]Although not used in the previous example, a similar remark regarding the (in-)significance of last job stages in deadlock avoidance applies to the implementation of RUN DAP.

Then, part 3 of Definition 10 implies that this implementation of RO on the considered RAS imposes the following set of linear inequalities on the system state:

$$\begin{bmatrix} 1 & & & & 1 \\ 1 & & 1 & & 1 \\ 1 & & & 1 & 1 \\ & 1 & 1 & & \\ & 1 & & 1 & \\ & & 1 & 1 & \end{bmatrix} \mathrm{s} \leq \begin{bmatrix} 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{11}$$

In Equation 11, each inequality corresponds to a pair (R_i, R_j) with $R_i < R_j$, and with all these pairs ordered lexicographically in increasing order (i.e., (R_1, R_2), (R_1, R_3), etc.). □

The policy correctness is proven in [43]. Given a certain FMS configuration, RO, like RUN, essentially defines a *family* of DAP's, generated by all the possible *total* orderings imposed on the FMS resources. The computation required for the initial setup of the policy (i.e., classification of the different route stages to left, right and undirected) is of complexity $O(D)$, where D is the total number of distinct processing stages executed in the considered RAS, while the number of constraints on the RAS state to be checked on-line, is of order $O(m^2)$, where m is the number of system resources. Therefore, the policy is *scalable*.

5.3 Ordered States and the FMS Banker's Algorithm

The classical Banker's algorithm [26] is based on the observation that a state is safe if its running processes can be ordered in such a manner that each process in the ordering can terminate using only its currently allocated resources, resources currently available in the system, and also, resources currently allocated to processes which are preceding it in the order. In the context of correct and scalable FMS DAP's, this idea leads to the concept of the *ordered state*, defined as follows:

Definition 11 *[38] Let $D = \sum_{j=1}^{n} l(j)$, the number of distinct route stages supported by a given system. RAS state* s(t) *is* ordered *iff there exists an ordering of the set of distinct job stages, $o() : \{JT_{jk} : \ j = 1, \ldots, n, \ k = 1, \ldots, l(j)\} \rightarrow \{1, \ldots, D\}$, such that the resource requirements for processing to completion a job instance j_i in stage $JT^{(i)}$ can be satisfied by means of the free resources in state* s(t)*, plus the resources held by job instances j_q in stages $JT^{(q)}$, with $q \leq i$.*

Let $\mathcal{S}_o$ denote the set of ordered RAS states. In [38], it is shown that $\mathcal{S}_o$ is a strongly connected subspace of the RAS STD, containing the empty state, and therefore, the restriction of the system

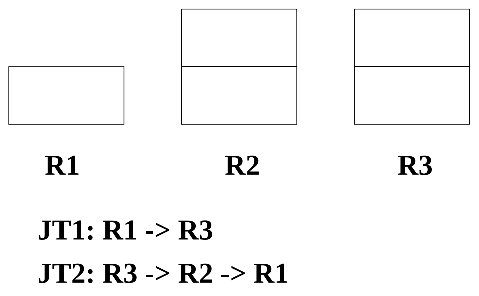


Figure 13: Example: A Single-Unit RAS for Banker's algorithm demonstration

operation on this set defines a correct DAP.

Definition 11 provides also immediately an algorithm for testing whether a state is ordered or not:

Definition 12 – FMS Banker's Algorithm *[38]*

```
1. Set $\mathcal{UJ} := \{JT_{jk},\ j = 1, \ldots, n,\ k = 1, \ldots, l(j)\}$; $i = 0$; ORDERED:=TRUE.
2. Repeat
    (a) $i := i + 1$;
    (b) Try to find a job stage $JT_{jk} \in \mathcal{UJ}$ the instances of which can terminate by using their
        currently allocated resources, plus the currently free resource units.
    (c) If no such a job stage can be found, ORDERED:= FALSE.
    (d) else $JT_{jk} \equiv JT^{(i)}$; $\mathcal{UJ} := \mathcal{UJ} \setminus \{JT_{jk}\}$; release the resources held by the job instances of
        $JT_{jk}$ to the pool of the free resource units.
   until $(\mathcal{UJ} = \emptyset) \vee$ (ORDERED=FALSE)
3. If (ORDERED=FALSE) return FALSE else return TRUE
```

Example Consider the Single-Unit RAS of Figure 13. This system consists of three resources, R_1, R_2, R_3, with $C_1 = 1$ and $C_2 = C_3 = 2$. In its current configuration, the system supports two job types: $JT_1 =< R_1,\ R_3 >$ and $JT_2 =< R_3,\ R_2,\ R_1 >$. The reader should be able to verify that state s $=< 1\ 0\ 1\ 1\ 0 >$ is ordered, with a valid ordering being $JT^{(1)} = JT_{11}$, $JT^{(2)} = JT_{22}$ and $JT^{(3)} = JT_{21}$. On the other hand, state s$' =< 1\ 0\ 2\ 1\ 0 >$ is not ordered, even though it is safe:

advancing one instance of job stage JT_{21} to its next stage allows the job instance in stage JT_{11} to run to completion, which further allows the remaining jobs to finish. $\square$

It is the inability of Banker's logic to discern the viability of the partial job advancements demonstrated in the previous example that renders the algorithm suboptimal. This is the price paid to keep the algorithm complexity polynomial in the system size. Indeed, similar to the "classical" Banker's Algorithm, the correctness of the FMS Banker's results from the fact that the re-usability of the system resources implies a monotonic increase of the pool of free resources whenever job instances terminate, and makes backtracking unnecessary during the search for a feasible job stage ordering. Hence, the complexity of the above algorithm is polynomial, specifically, $O(mD \log D)$.

6 Efficiency Considerations for Polynomial-Kernel DAP's

It has already been observed that Polynomial-Kernel policies attain their tractability at the cost of suboptimality. Hence, a valid point for this line of research is to try to reduce suboptimality as much as possible. Before, however, we start considering this problem, we must define a *metric* for measuring the efficiency of these policies. There are two general approaches to establish such a metric. The first approach compares the attainable system performance under the control of various DAP's with respect to typical performance measures like process throughput(s), waiting times, Work-In-Process, and resource utilizations. Results along these lines for RUN and RO DAP's can be found in [48, 51, 53]. In this chapter, we shall focus on the second approach, which evaluates the efficiency of the different DAP's, based on the concept of *operational flexibility*. Specifically, the flexibility allowed by the evaluated policy is compared to the flexibility attained by the maximally permissive (optimal) DAP, by assessing the coverability of the safe space $\mathcal{S}_s$ by the policy-admissible subspace, $\mathcal{S}(\mathcal{P})$. More formally, consider the Polynomial-Kernel policy defined by property $\mathcal{H}$, and let $\mathcal{S}(\mathcal{H}) \equiv \{s^i \in \mathcal{S} : \mathcal{H}(s^i) \text{ is TRUE}\}$, i.e., the policy admissible subspace. Then, a viable policy efficiency measure is provided by the ratio

$$I = \frac{|\mathcal{S}(\mathcal{H})|}{|\mathcal{S}_s|} \tag{12}$$

where $|\mathcal{S}|$ denotes the cardinality number of set $\mathcal{S}$.

Due to the typically large size of the $\mathcal{S}(\mathcal{H})$ and $\mathcal{S}_s$ subspaces, their explicit enumeration will not be possible, and therefore we must resolve to simulation and statistical sampling techniques. Such a technique, known as the *Co-space Simulation* technique, is developed in [39]. Briefly, this approach recognizes that the set of safe states of a given SU-RAS, Q, corresponds to the reachability set of a "co-system", Q', which is defined from the original RAS Q, by reversing its job routes. Hence, the operation of the co-system Q' is simulated until a sufficiently large sample of states is obtained. According to the previous remark, this sample set consists of safe states of the original system. In continuation, the condition $\mathcal{H}$ defining the evaluated DAP is applied on the extracted sample set, and the portion of the sample states admitted by the policy is determined. This portion expresses the policy coverability of the extracted sample set, and constitutes a point estimate for index I. Application of this technique to the Polynomial-Kernel DAP's of Section 5, and experimental evaluation results can be found in [39, 43, 38]. In the rest of this section,we discuss some properties of Polynomial-Kernel DAP's which can be used to enhance the operational flexibility of these policies, when implemented on any given FMS configuration.

6.1 Policy disjunctions and essential difference

The first way to improve the efficiency of an FMS structural controller employing Polynomial-Kernel DAP's, w.r.t. the metric of Equation 12, is based on the following proposition:

Proposition 4 *Given two conditions $\mathcal{H}_1()$ and $\mathcal{H}_2()$ defining correct Polynomial-Kernel DAP's, the policy defined by the* disjunction $\mathcal{H}_1() \vee \mathcal{H}_2()$ *is another correct Polynomial-Kernel DAP.*

To see this, simply notice that acceptance of a state s by the policy disjunction implies that at least one of the two policy defining conditions, $\mathcal{H}_1()$, $\mathcal{H}_2()$, evaluates to TRUE at s, and therefore, state s is safe. Furthermore, if state $\mathrm{s} \in \mathcal{S}(\mathcal{H}_i),\ i \in \{1,2\}$, then the correctness of the corresponding policy implies the existence of at least one feasible event e, which is enabled by that policy, and $\delta(e,\mathrm{s}) \equiv \mathrm{s}' \in \mathcal{S}(\mathcal{H}_i)$ (cf. Theorem 2). But then, $\mathrm{s}' \in \mathcal{S}(\mathcal{H}_1 \vee \mathcal{H}_2)$, and according to Theorem 2, the policy defined by $\mathcal{H}_1() \vee \mathcal{H}_2()$ is correct.

It is also easy to see that the subspace admitted by the policy disjunction is the *union* of the subspaces admitted by the two constituent policies. If it happens that:

$$(\mathcal{S}(\mathcal{H}_1) \not\subseteq \mathcal{S}(\mathcal{H}_2)) \wedge (\mathcal{S}(\mathcal{H}_2) \not\subseteq \mathcal{S}(\mathcal{H}_1)) \tag{13}$$

then $\mathcal{S}(\mathcal{H}_1) \cup \mathcal{S}(\mathcal{H}_2)$ is richer in states than any of its constituents. Therefore, the resulting policy is more efficient with respect to index I. Two polynomial-kernel policies based on conditions $\mathcal{H}_1$ and $\mathcal{H}_2$ that satisfy Equation 13, are characterized as *essentially different*. Moreover, the policy defined by their disjunction is more efficient than the policy defined by each one of them, as it was just shown.

The essential difference of the Polynomial-Kernel policies presented in Section 5 is analyzed in [38]. It turns out that RUN and the FMS Banker's algorithm are essentially different, while RO is subsumed by Banker's.

6.2 Optimal and orthogonal orderings for RUN and RO DAP's

A second opportunity for improving the efficiency of RUN and RO DAP's is provided by the fact that the defining logic of these two policies essentially leads to entire families of policies for a given FMS configuration. As we saw in Section 5, each member of these families is defined by a distinct ordering of the system resource set. Hence, a naturally arising question is which of these orderings leads to the most efficient policy implementation. In this way, an *optimization* problem is defined, which can be considered as a *parameter-tuning* (optimization) problem. This optimization problem has been characterized as the *"Optimal Ordering"* problem, and some further discussion on its formulation and solution can be found in [39, 43].

Furthermore, the aforestated richness of RUN and RO implementations on a given RAS configuration raises the possibility of the existence of different orderings within the same policy family which lead to the admissibility of subspaces complementary to each other. This idea, combined with the closure of Polynomial-Kernel DAP's with respect to policy disjunction, has led to the definition of the *"Orthogonal Ordering"* problem. Details of the problem formulation and its solution can be found in [48].

6.3 Combining Polynomial-Kernel DAP's with partial search

A last idea that can lead to efficiency improvement of Polynomial-Kernel DAP's, is to expand the state space admitted by such a policy, by allowing transitions to states s which fail to satisfy the policy defining condition $\mathcal{H}()$, – i.e., $\mathcal{H}(\mathrm{s}) = \mathrm{FALSE}$ – but for which their inclusion in the target set $\mathcal{S}_s$ can be established through *controlled partial search*, i.e., n-step *look-ahead* schemes. Specifically, for an n-step look-ahead scheme, state s with $\mathcal{H}(\mathrm{s}) = \mathrm{FALSE}$ is admissible, if there exists a sequence of feasible events, w, such that (i) $|w| \leq n$, and (ii) $\delta(w, \mathrm{s}) = \mathrm{s}' \in \mathcal{S}(\mathcal{H})$. Since this new admissibility condition is of *existential* character, it is deemed that it can increase the policy efficiency with rather small computational cost, for reasonable sizes of look-ahead horizons.

It is interesting to notice how the length, n, of the look-ahead horizon partitions the target set $\mathcal{S}_s$ accepted by the optimal DAP: $n = 0$ defines the *kernel* set $\mathcal{S}(\mathcal{H})$, while every time that the look-ahead horizon is increased by one step, say from n to $n+1$, an additional *ring* of states is added

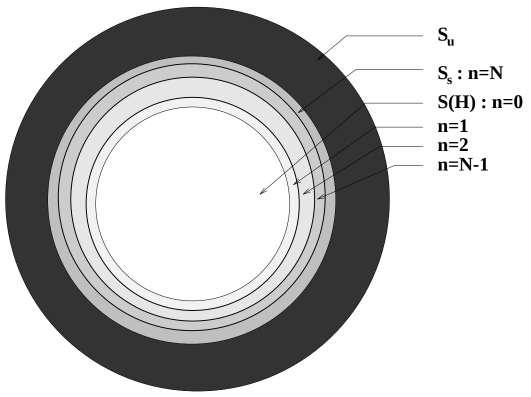


Figure 14: Expanding the admitted subspace of Polynomial-Kernel DAP's through n-step lookahead search

to the set of states admitted when the horizon length is equal to n. Obviously, for finite state spaces this expansion continues only up to the point that the entire set $\mathcal{S}_s$ is covered, for some maximal length N. This partitioning of the optimal subspace on the basis of the lookahead horizon size, n, is depicted in Figure 14.

It might also be efficient to store some of the results of these partial searches in a *look-up table*. Specifically, some states s for which $\mathcal{H}(\mathrm{s}) =$ FALSE, but which have been shown to belong to $\mathcal{S}_s$ through the search mechanism, can be stored to a table for future reference. In this way, the cost of look-ahead search is spared for these states. However, given the large size of the underlying spaces, storing the entire set of states found to be admissible through n-step look-ahead search, in general, will not be practically possible. All the same, the idea can become practical by exploiting the following economies introduced by the system structure and operation:

i. From the FMS structural analysis of Section 3, it easily follows that:

$$\forall \mathrm{s}^1,\ \mathrm{s}^2 \text{ s.t } \forall i,\ \mathrm{s}_i^1 \leq \mathrm{s}_i^2 : \ \mathtt{safe}(\mathrm{s}^2) \Rightarrow \ \mathtt{safe}(\mathrm{s}^1) \tag{14}$$

This result implies a *covering* relationship over the set of safe states admitted by a look-ahead policy, in the sense that an admissible state should be stored in the policy look-up table, only if its admissibility is not implied by an already stored state and Equation 14.

ii. Sometimes, the system operation presents considerable *localization* with respect to its underlying state space, i.e., a rather small number of states is revisited a significant number of times. Then, an appropriately sized look-up table can function as a *cache memory* which holds information about the structure of the currently working subspace. Furthermore, a *forgetting* mechanism is required to update the set of stored states as the system "drifts" to new state regions. Such a mechanism can be provided, for instance, by time-stamping the stored states with the last time that they were visited during the system operation – a variation of a scheme known as *aging* [14].

An application of the combination of Polynomial-Kernel deadlock avoidance policies with partial search can be found in [38].

7 Additional Issues and Future Directions in FMS Structural Control

The starting point of this chapter has been the observation that current trends in discrete-part manufacturing require extensive levels of automation of the underlying shop-floor activity, and in order to support this extensive automation, a more sophisticated real-time control methodology for these environments is needed. Specifically, the extensive automation, or even autonomy, required for the operation of these systems gives rise to a new set of of problems concerning the establishment of their logically correct and robust behavior, which is collectively known as the structural control problem of contemporary FMS. In the rest of the chapter, we focused on one particular problem of this area, the resolution of the manufacturing system deadlock, which has been a predominant FMS structural control issue. The details of the FMS deadlock problem and its solution depend on the operational characterisics of the system, as it was revealed by Section 2.1 and the RAS taxonomy of Section 2.2. Restricting the subsequent discussion to the most typical case where deadlocks arise due to ineffective allocation of the system buffering capacity (i.e., the first class of the underlying RAS taxonomy), we were able to show that, even though, in the general case, the problem of optimal (maximally permissive) deadlock avoidance is NP-hard, when the system is configured so that every resource (i.e., workstation and/or MHS-component) can stage at least two parts at a time, we can obtain the optimal DAP with polynomial computational cost. Hence, the problem of FMS deadlock due to finite buffering capacity is conclusively resolved, for all practical purposes. Furthermore, to completely cover the class of Single-Unit RAS, we presented Polynomial-Kernel DAP's, a class of suboptimal yet efficient DAP's which are also computationally tractable.

In the rest of this section we briefly discuss some other aspects of this work, not covered in the material of this chapter. For one thing, it has already been mentioned that the Polynomial-Kernel DAP's of Section 5 have been extended to cover the Single-Type and Conjuctive RAS of the aforestated taxonomy. These results are straightforward generalizations of the policies presented in Section 5, and can be found in [55, 54]. Presently, work is under devlopment addressing the last class of the presented RAS taxonomy, i.e., the integration of deadlock avoidance with flexible job routing. This problem is non-trivial, since, in addition to the increased complexity associated with the underlying state-safety decision problem, the introduction of routing flexibility results in very high *space* complexity due to the exponentially large number of routing options associated with each job type. Some initial results investigating the trade-off between increased computational effort and the benefits of routing flexibility are presented in [42]. A different approach on the issue is presented in [49]: acknowledging the increased complexity of the on-line job rerouting problem, the author suggests the exploitation of the system inherent flexibilities for effective accommodation job of different operational contingencies, like machine breakdowns and the arrival of expedient jobs. So, this work essentially deals with the effective reconfiguration of the system in face of a major disruption, and "bridges" the existing results on deadlock avoidance with the second main area of FMS structural control, i.e., the design of protocols for exception handling and graceful degradation. Finally, additional directions of extending the developed DAP's to new RAS classes include the study of hierarchically structured RAS, and RAS in which the length of the different job routes is not predetermined, but depends on the occurrence of various events. These results are currently under development.

FMS structural control and the developed policies have also significant repercussion on the various aspects of the system performance. In a sense, the constraints imposed by a structural control policy define the feasibility space for any performance-optimizing model. In other words, they significantly shape the system *capacity*.[9] As it is observed in [23], *"defining, measuring and respecting capacity are important at all levels of the [system] hierarchy. No system can produce outside its capacity, and it is futile at best, and damaging, at worst, to try ... It is essential therefore to determine what capacity is, then to develop a discipline for staying within it."*. Initial results for scheduling the capacity of the structurally controlled FMS are presented in [48, 51, 53]. An architecture for real-time integration of structural and performance-oriented control in contemporary FMS controllers is presented in [48]. Essentially, the SC policy, at any point of the system operation, censors the decisions of the system dispatcher for action admissibility with respect to safety. In some more recent work [50], the effect of this censorship on the stability[10] of some well-known distributed scheduling policies has been

[9]For a rigorous definition of this concept, see [22].

[10]Practically, a DAP is stable if it can provide the maximum throughput attainable over the entire class of scheduling

analyzed. It turns out that even when maximally permissive deadlock avoidance control is applied, most of the distributed policies which have been formally shown to be stable under the assumption of infinite capacitated buffers, are not optimal (i.e., stable) any more. This result reiterates the significance of adequate modeling of the system structural aspects while analyzing its performance, and reintroduces the scheduling problem in this new environment. In a similar fashion, the entire FMS tactical planning problem (cf. Introduction) needs to be re-considered.

A last contribution of this work, and a new horizon for additional research, is the systematic treatment of complexity in large-scale supervisory control problems. As it was stated in the Introduction, current supervisory control theory has a very significant impact on the formulation and the rigorous analysis of the solvability (decidability) of problems rising in the area of logical control of Discrete Event Systems [47]. However, most of the proposed algorithms, in their effort to remain generic enough, address these problems at a (formal lagnuage-theoretic) syntactical level, which ignores the detailed structure of the controlled system. As a result, these solutions are of complexity polynomial to the size of the underlying system state space, and therefore, intractable for the cases where this size explodes badly (as in the case of the deadlock avoidance problem). The work presented herein demonstrates that much can be gained in terms of mastering this complexity, if the undertaken analysis focuses also on the specifics of the system behavior (language *semantics*). It is our feeling that there are a lot of DES logical control problems currently begging computationally efficient (real-time) solution. Understanding the problem attributes determining the problem complexity and establishing rational trade-offs between computational tractability and operational efficiency are very important but also challenging problems for the field.

References

[1] A survey on manufacturing technology. *The Economist*, March 5th:3–18, 1994.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[3] T. Araki, Y. Sugiyama, and T. Kasami. Complexity of the deadlock avoidance problem. In *2nd IBM Symp. on Mathematical Foundations of Computer Science*, pages 229–257. IBM, 1977.

policies.

[4] G. Arango and R. Prieto-Diaz. Domain analysis concepts and research directions. In R. Prieto-Diaz and G. Arango, editors, *Domain Analysis and Software Systems Modeling*, pages 9–33. IEEE Computer Society Press, 1991.

[5] Z. A. Banaszak and B. H. Krogh. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Trans. on Robotics and Automation*, 6:724–734, 1990.

[6] D. A. Bodner and S. A. Reveliotis. Virtual factories: An object-oriented simulation-based framework for real-time fms control. In *ETFA'97*, pages 208–213. IEEE, 1997.

[7] R. K. Boel, L. B. Naoum, and V. V. Breusegem. On forbidden state problems for a class of controlled petri nets. *IEEE Trans. on Automatic Control*, 40:1717–1731, 1995.

[8] H. Cho, T. K. Kumaran, and R. A. Wysk. Graph-theoretic deadlock detection and resolution for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 11:413–421, 1995.

[9] H. C. Co, J. S. Biermann, and S. K. Chen. A methodical approach to the flexible manufacturing-system batching, loading and tool configuration problems. *Int. J. Prod. Res.*, 28:2171–2186, 1990.

[10] E. G. Coffman, M. J. Elphick, and A. Shoshani. System deadlocks. *Computing Surveys*, 3:67–78, 1971.

[11] D. Connors, G. Feigin, and D. Yao. Scheduling semiconductor lines using a fluid network model. *IEEE Trans. on Robotics & Automation*, 10:88–98, 1994.

[12] National Research Council. Information technology and manufacturing. Technical report, National Academy Press, 1993.

[13] J. G. Dai, D. H. Yeh, and C. Zhou. The qnet method for re-entrant queueing networks with priority disciplines. *Op. Res.*, 45:610–623, 1997.

[14] H. M. Deitel. *Operating Systems.* Addison Wesley, Reading, MA, 1990.

[15] E. W. Dijkstra. Cooperating sequential processes. Technical report, Technological University, Eindhoven, Netherlands, 1965.

[16] P. F. Drucker. The emerging theory of manufacturing. *Harvard Business Review*, May-June:94–102, 1990.

[17] J. Ezpeleta, J. M. Colom, and J. Martinez. A petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. on R&A*, 11:173–184, 1995.

[18] M. P. Fanti, B. Maione, S. Mascolo, and B. Turchiano. Event-based feedback control for deadlock avoidance in flexible production systems. *IEEE Trans. on Robotics and Automation*, 13:347–363, 1997.

[19] L. Ferrarini, M. Narduzzi, and M. Tassan-Solet. A new approach to modular liveness analysis conceived for large logic controllers' design. *IEEE Trans. on Robotics and Automation*, 10:169–184, 1994.

[20] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co., New York, NY, 1979.

[21] S. B. Gershwin. Hierarchical flow control: A framework for scheduling and planning discrete events in manufacturing systems. *Proceedings of the IEEE*, 77:195–209, 1989.

[22] S. B. Gershwin. *Manufacturing Systems Engineering.* PTR Prentice Hall, Englewood Cliffs, N.J., 1994.

[23] S. B. Gershwin, R. R. Hildebrant, R. Suri, and S. K. Mitter. A control perspective on recent trends in manufacturing systems. *IEEE Control Systems Magazine*, 6:3–15, 1986.

[24] A. Giua and F. DiCesare. Petri net structural analysis for supervisory control. *IEEE Trans. on Robotics and Automation*, 10:169–184, 1994.

[25] E. M. Gold. Deadlock prediction: Easy and difficult cases. *SIAM Journal of Computing*, 7:320–336, 1978.

[26] A. N. Habermann. Prevention of system deadlocks. *Comm. ACM*, 12:373–377, 1969.

[27] J. W. Havender. Avoiding deadlock in multi-tasking systems. *IBM Systems Journal*, 2:74–84, 1968.

[28] R. D. Holt. Some deadlock properties of computer systems. *ACM Computing Surveys*, 4:179–196, 1972.

[29] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA, 1979.

[30] F. S. Hsieh and S. C. Chang. Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 10:196–209, 1994.

[31] M. D. Jen and F. DiCesare. Synthesis using resource control nets for modeling shared-resource systems. *IEEE Trans. on Robotics and Automation*, 11:317–327, 1995.

[32] S. B. Joshi, E. G. Mettala, J. S. Smith, and R. A. Wysk. Formal models for control of flexible manufacturing cells: Physical and system models. *IEEE Trans. on Robotics and Automation*, 11:558–570, 1995.

[33] P. Kumar, K. Kothandaraman, and P. Ferreira. Scalable and maximally-permissive deadlock avoidance for fms. In *ICRA'98 (submitted).* IEEE R&A, 1998.

[34] P. R. Kumar. Scheduling manufacturing systems of re-entrant lines. In D. D. Yao, editor, *Stochastic Modeling and Analysis of Manufacturing Systems*, pages 325–360. Springer-Verlag, 1994.

[35] P. R. Kumar and S. P. Meyn. Stability of queueing networks and scheduling policies. *IEEE Trans. Autom. Control*, 40:251–260, 1995.

[36] P. R. Kumar and S. P. Meyn. Duality and linear programs for stability and performance analysis of queueing networks and scheduling policies. *IEEE Trans. Autom. Control*, 41:4–17, 1996.

[37] P. R. Kumar and T. I. Seidman. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Trans. Autom. Control*, 35:289–298, 1990.

[38] M. Lawley, S. Reveliotis, and P. Ferreira. The application and evaluation of banker's algorithm for deadlock-free buffer space allocation in flexible manufacturing systems. *Intl. Jrnl. of Flexible Manufacturing Systems (to appear).*

[39] M. Lawley, S. Reveliotis, and P. Ferreira. Fms structural control and the neighborhood policy, part 2: Generalization, optimization and efficiency. *IIE Trans.*, 29:889–899, 1996.

[40] M. Lawley, S. Reveliotis, and P. Ferreira. Design guidelines for deadlock handling strategies in flexible manufacturing systems. *Intl. Jrnl. of Flexible Manufacturing Systems*, 9:5–29, 1997.

[41] M. A. Lawley. *Structural Analysis & Control of Flexible Manufacturing Systems.* PhD thesis, University of Illinois, Urbana, IL, 1995.

[42] M. A. Lawley. A control model for integrating routing flexibility with algebraic deadlock avoidance in flexible manufacturing systems. Technical Report Res. Mem. 97-12 (submitted to Intl. Jrnl of Prod. Res.), School of Industrial Eng., Purdue Univ., 1997.

[43] M. A. Lawley, S. Reveliotis, and P. Ferreira. The resource order policy: A configurable and scalable control policy for deadlock-free buffer-space allocation in flexible manufacturing systems. Technical Report UILU-ENG 96-4014 (submitted to the IEEE Trans. on Robotics and Automation), University of Illinois at Urbana-Champaign, 1996.

[44] S. H. Lu and P. R. Kumar. Distributed scheduling based on due dates and buffer priorities. *IEEE Trans. on Aut. Control*, 36:1406–1416, 1991.

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

[46] J. L. Peterson. *Operating System Concepts.* Addison-Wesley, 1981.

[47] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[48] S. A. Reveliotis. *Structural Analysis & Control of Flexible Manufacturing Systems with a Performance Perspective.* PhD thesis, University of Illinois, Urbana, IL, 1996.

[49] S. A. Reveliotis. Accommodating fms operational contingencies through routing flexibility. Technical Report (submitted to IEEE Trans. on R&A), School of Industrial & Systems Eng., Georgia Tech, 1997.

[50] S. A. Reveliotis. The destabilizing effect of blocking due to finite buffering calacity in multi-class queueing networks. Technical Report (submitted to IEEE Trans. on Autom. Control), School of Industrial & Systems Eng., Georgia Tech, 1997.

[51] S. A. Reveliotis and P. M. Ferreira. An analytical framework for evaluating and optimizing the performance of structurally controlled fms. In *1996 IEEE International Conference on Robotics and Automation*, pages 864–869. IEEE Robotics and Automation Society, 1996.

[52] S. A. Reveliotis and P. M. Ferreira. Deadlock avoidance policies for automated manufacturing cells. *IEEE Trans. on Robotics & Automation*, 12:845–857, 1996.

[53] S. A. Reveliotis and P. M. Ferreira. Performance evaluation of structurally controlled fms: Exact and approximate approaches. In *Flexible Automation and Intelligent Manufacturing, 1996*, pages 829–838. Manufacturing Research Center, Georgia Tech., 1996.

[54] S. A. Reveliotis and M. A. Lawley. Efficient implementations of banker's algorithm for deadlock avoidance in flexible manufacturing systems. In *ETFA '97*, pages 214–220. IEEE, 1997.

[55] S. A. Reveliotis, M. A. Lawley, and P. M. Ferreira. Polynomial complexity deadlock avoidance policies for sequential resource allocation systems. *IEEE Trans. on Automatic Control*, 42:1344–1357, 1997.

[56] F. A. Rodammer and P. Jr. White. A recent survey of production scheduling. *IEEE Trans. on SMC*, 18:841–851, 1988.

[57] A. K. Sethi and S. P. Sethi. Flexibility in manufacturing: A survey. *The International Journal of Flexible Manufacturing Systems*, 2:289–328, 1989.

[58] A. Sharifnia. Stability and performance of distributed production control methods based on continuous-flow models. *IEEE Trans Autom. Control*, 39:725–737, 1994.

[59] R. S. Sreenivas. On a weaker notion of controllability of a language k with respect to a language l. *IEEE Trans. on Automatic Control*, 38:1446–1447, 1993.

[60] R. S. Sreenivas. On the existence of finite state supervisors for arbitrary supervisory control problems. *IEEE Trans. on Automatic Control*, 39:856–861, 1994.

[61] B. Srivastava and W-H Chen. Heuristic solutions for loading in flexible manufacturing systems. *IEEE Trans. on R&A*, 12:858–868, 1996.

[62] K. E. Stecke. Design, planning, scheduling and control problems of flexible manufacturing systems. *Annals of Operations Research*, 3, 1985.

[63] K. E. Stecke and N. Raman. Production planning decisions in flexible manufacturing systems with random material flows. *IIE Trans.*, 26:2–17, 1994.

[64] F. F. Suarez, M. A. Cusumano, and C. H. Fine. An empirical study of manufacturing flexibility in printed circuit board assembly. *Operations Research*, 44:223–240, 1997.

[65] N. Viswanadham, Y. Narahari, and T. L. Johnson. Deadlock avoidance in flexible manufacturing systems using petri net models. *IEEE Trans. on Robotics and Automation*, 6:713–722, 1990.

[66] W. L. Winston. *Introduction To Mathematical Programming: Applications and Algorithms, 2nd ed.* Duxbury Press, Belmont, CA, 1995.

[67] R. A. Wysk, N. S. Yang, and S. Joshi. Detection of deadlocks in flexible manufacturing cells. *IEEE Trans. on Robotics and Automation*, 7:853–859, 1991.

[68] R. A. Wysk, N. S. Yang, and S. Joshi. Resolution of deadlocks in flexible manufacturing systems: Avoidance and recovery approaches. *Journal of Manufacturing Systems*, 13:128–138, 1994.

[69] K. Y. Xing, B. S. Hu, and H. X. Chen. Deadlock avoidance policy for petri net modeling of flexible manufacturing systems with shared resources. *IEEE Trans. on Aut. Control*, 41:289–295, 1996.

[70] M. Zhou and F. Dicesare. Parallel and sequential mutual exclusions for petri net modeling of manufacturing systems with shared resources. *IEEE Trans. on Robotics and Automation*, 7:515–527, 1991.