

Chapter 2

LOGICAL CONTROL OF DISJUNCTIVE / CONJUNCTIVE RESOURCE ALLOCATION SYSTEMS

In this chapter we undertake the systematic investigation of the RAS logical control problem, that was introduced in Sections 2 and 3 of Chapter 1. However, at this first stage, our analysis will be confined to the behavioral context of the *Disjunctive / Conjunctive (DIS-CON-)RAS* sub-class, that was defined in the taxonomy of Section 2.2, in Chapter 1. We remind the reader that this RAS sub-class allows for arbitrary resource allocation requests and process routing flexibility, but it does not allow for process parallelization. Hence, at every point in time, every active process instance in the resource allocation system constitutes a single atomic entity, executing one of its processing stages. The extension of the relevant theory in order to encompass more complex RAS behaviors, involving (sub-)process coordination, requires a more sophisticated modelling framework and more complicated analysis tools, and it is deferred to Chapter 5. In the sequel, any invocation of the RAS concept should be considered in the context of the DIS-CON-RAS sub-class, unless otherwise specified.

According to the discussion of Section 2.2, in Chapter 1, the class of DIS-CON-RAS is obtained from Definition 1 by requiring that, for each process type Π_j , the corresponding data structure \mathcal{G}_j is an *acyclic graph* with node set equal to the set of processing stages, \mathcal{S}_j . Each edge (Ξ_{jk}, Ξ_{jq}) of this graph implies that processing stage Ξ_{jq} can be an *immediate successor* stage for processing stage Ξ_{jk} (or, equivalently, that processing stage Ξ_{jk} can be an *immediate predecessor* stage for processing stage Ξ_{jq}). Furthermore, we shall use the notation \mathcal{S}_j^{\nearrow} (resp., \mathcal{S}_j^{\searrow}) in order to refer to the set of stages that correspond to *source* (resp., *sink*) nodes of \mathcal{G}_j . Every path from a "source" stage $\Xi_{jk} \in \mathcal{S}_j^{\nearrow}$ to a "sink" stage $\Xi_{jq} \in \mathcal{S}_j^{\searrow}$ corresponds to a potential "*process plan*" for process type Π_j .

For this simpler class of systems, we employ the formal framework of *Finite State Automata (FSA)* (Hopcroft and Ullman, 1979) in order to provide a natural yet rigorous characterization of the RAS behavior, and of the concepts of *safety* and *maximally permissive nonblocking supervision*, that were informally introduced in the discussion of Chapter 1. The rigorous characterization of the considered logical control problem subsequently enables the systematic study of its computational complexity. It turns out that the problem belongs to the notorious class of NP-hard problems (Garey and Johnson, 1979), and this result establishes a trade-off between the permissiveness of the proposed non-blocking SC policies and their computational tractability. Hence, in the third part of this chapter we outline a general methodology for resolving this trade-off; detailed implementation of the presented ideas in the context of various RAS sub-classes will be provided in Chapters 4 and 5. The chapter concludes with the discussion of some variations of the RAS logical control problem that incorporate (i) potentially uncontrollable elements in the RAS behavior, and (ii) operational contingencies that will necessitate the re-design of the applied supervisory control policy.

1. Finite State Automaton-based modelling of the RAS behavior

As it was explained in the introductory chapter, logical control focuses on the *logical* or *qualitative* properties of the RAS dynamics, and it is concerned with the *logical sequencing* of the various resource allocation - related events taking place in the system. Furthermore, it was argued in that chapter that any approach trying to enforce a particular event sequence in the system behavior by controlling the exact timing of the occurrence of the various events of interest, would be too *brittle* in the considered application contexts, due to the stochasticity of the sojourn times associated with the various processing stages. Hence, this part of our analysis will ignore time completely, and it will employ only *untimed* models providing qualitative yet formal characterizations of the RAS behavior.

1.1 Finite State Automata: Basic Concepts and Definitions

Among the class of qualitative behavioral models employed by Discrete Event System theory, the most straightforward, and, probably, the most widely used, is the *Finite State Automaton (FSA)* (Hopcroft and Ullman, 1979; Cassandras and Lafortune, 1999). A formal definition of this model is as follows:

DEFINITION 2 (Cassandras and Lafortune, 1999) A (Deterministic) Finite State Automaton (FSA) G is a 6-tuple

$$G = \langle E, S, f, \Gamma, s_0, S_m \rangle$$

where

- E is a finite set, called the event set of the automaton;
- S is a finite set, called the state set of the automaton;
- $f : S \times E \rightarrow S$, is the state transition function, i.e., $\forall s \in S, \forall e \in E$, $f(s, e) = s'$ means that there is a transition from state s to state s' that is triggered by event e ; in general, f is a partial function on its domain, i.e., certain events cannot occur in state s ;
- $\Gamma : S \rightarrow 2^E$ is the feasible event function, i.e., $\forall s \in S$, $\Gamma(s)$ denotes the set of all events e for which $f(s, e)$ is defined;
- $s_0 \in S$ is the initial state of the automaton;
- $S_m \subseteq S$ is the set of marked states.

The transitional structure expressed by the FSA model can be visualized by a labelled graph \mathcal{G} ; this graph is known as the *state transition diagram (STD)* of the FSA, and its node set corresponds to the state set S of the automaton, its edge set is defined by the state transition function, and its edge label set corresponds to the event set E of the automaton.

It is obvious from the above definitions that the FSA and its corresponding STD can be perceived as a complete map for the behavioral evolution of the modelled system. This effect can be formalized as follows:

DEFINITION 3 (Cassandras and Lafortune, 1999) Consider an FSA $G = \langle E, S, f, \Gamma, s_0, S_m \rangle$ and let E^* denote the set containing all the finite-length sequences that can be generated from E , including the empty sequence ϵ .

1 The FSA state transition function f is naturally extended to $S \times E^*$ as follows:

$$\forall s \in S, \quad f(s, \epsilon) \equiv s \quad (2.1)$$

$$\forall s \in S, \forall u \in E^*, \forall e \in E, \quad f(s, ue) \equiv f(f(s, u), e) \quad (2.2)$$

2 The language $\mathcal{L}(G)$ generated by G is defined by

$$\mathcal{L}(G) \equiv \{u \in E^* : f(s_0, u)!\}^1 \quad (2.3)$$

¹i.e., $f(s_0, u)$ involves only transitions corresponding to feasible events, and therefore, it is well-defined

3 The language $\mathcal{L}_m(G)$ marked by G is defined by

$$\mathcal{L}_m(G) \equiv \{u \in \mathcal{L}(G) : f(s_0, u) \in S_m\} \quad (2.4)$$

In the STD context, $\mathcal{L}(G)$ can be described as the set of all event sequences $u \in E^*$ that can be traced on any path, not necessarily simple, starting from the initial state s_0 . $\mathcal{L}_m(G)$ is used to model event sequences that correspond to the achievement of some "milestone" in the system behavior.

1.2 FSA-based modelling of the RAS behavior

The formalism and concepts introduced in the previous section, provide all the necessary mathematical apparatus for developing an FSA-based characterization of the RAS behavior. We proceed to this characterization by providing first the formal definition of the RAS *state* that will be employed in the logical analysis of its behavior.

DEFINITION 4 Consider a RAS $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A}, \mathcal{T} \rangle$. For the purposes of logical analysis, its state $s(t)$ at time t is defined as a vector of dimensionality $D = \sum_{j=1}^n l(j)$ – i.e., equal to the total number of distinct processing stages in the system – and with components $s(q; t)$, $q = 1, \dots, D$, being in one-to-one correspondence with the RAS processing stages, Ξ_{jk} , $j = 1, \dots, n$, $k = 1, \dots, l(j)$. Furthermore, component $s(q(j, k); t)$, corresponding to processing stage Ξ_{jk} , indicates the number of process instances executing stage Ξ_{jk} at time t .

A natural way to define the correspondence between the state components and the RAS processing stages is by setting $q(j, k) = k + \sum_{r=1}^{j-1} l(r)$; this will be the mapping assumed in the following, unless otherwise stated. Also, to simplify the notation, in the following discussion we omit the dependence of state s on time t .

Notice that the information contained in the RAS state is sufficient for the determination of the distribution of the resource units to the various process stages, as well as of the *slack* (or *idle*) resource capacity in the system. In particular, we define the *slack* capacity, $\delta_i(s)$, of resource R_i at state s , by

$$\delta_i(s) \equiv C_i - \sum_{q=1}^D s(q(j, k)) \cdot A_{jk}(i) \quad (2.5)$$

Then, the set S of *feasible* resource allocation states for the considered RAS is defined by

$$S \equiv \{s \in (Z_0^+)^D : \delta_i(s) \geq 0, \forall i = 1, \dots, m\} \quad (2.6)$$

The finiteness of the resource capacities implies that $\text{card}(S) \equiv |S| < \infty$. However, in general, $|S|$ will be a *super-polynomial* function of the RAS size; in the particular case where each process stage requires a single unit from a single resource type, $|S| = \prod_{i=1}^m \frac{(C_i + |S(R_i)|)!}{C_i! |S(R_i)|!}$, where C_i denotes the capacity of resource R_i and $S(R_i)$ denotes the set of processing stages requesting resource R_i for their execution.

The set of *events*, E , that can change the system state, comprises: (i) the events e_{jk}^l , $j = 1, \dots, n$, $k \in \{1, \dots, l(j) : \Xi_{jk} \in \mathcal{S}_j^{\nearrow}\}$, corresponding to the *loading* of a new instance of process type Π_j into the system, that is to follow a process plan starting with stage Ξ_{jk} , (ii) the events e_{jkh}^a , $j = 1, \dots, n$, $k = 1, \dots, l(j)$, $h \in \{q : (\Xi_{jk}, \Xi_{jq}) \in \mathcal{G}_j\}$, corresponding to *advancement* of a process instance executing stage Ξ_{jk} to a successor stage Ξ_{jh} , and (iii) the events e_{jk}^u , $j = 1, \dots, n$, $k \in \{1, \dots, l(j) : \Xi_{jk} \in \mathcal{S}_j^{\searrow}\}$, corresponding to the *unloading* of a finished process instance of type Π_j , whose last processing stage was stage $\Xi_{jk} \in \mathcal{S}_j^{\searrow}$. Without loss of generality, it is assumed that, during a single state transition, only one of these events can take place. The resulting transition, however, is *feasible* only if the additionally requested set of resources can be obtained from the system slack capacity; i.e., for each state s , the set of *feasible* events, $\Gamma(s)$, contains only those events that abide to the resource allocation dynamics described in Section 2 of Chapter 1. Based on the above, the *state transition function* f and the *feasible event function* Γ for this automaton, are formally defined as follows:

$$\forall s \in S, \forall e \in E,$$

$$f(s, e) \equiv \begin{cases} s + \mathbf{1}_{q(j,k)} & \text{if } e \equiv e_{jk}^l \text{ and } s + \mathbf{1}_{q(j,k)} \in S \\ s - \mathbf{1}_{q(j,k)} + \mathbf{1}_{q(j,h)} & \text{if } e \equiv e_{jkh}^a \text{ and } s - \mathbf{1}_{q(j,k)} + \mathbf{1}_{q(j,h)} \in S \\ s - \mathbf{1}_{q(j,k)} & \text{if } e \equiv e_{jk}^u \text{ and } s - \mathbf{1}_{q(j,k)} \in S \\ \text{undefined} & \text{otherwise} \end{cases} \quad (2.7)$$

$\forall s,$

$$\Gamma(s) \equiv \{e \in E : f(s, e)!\} \quad (2.8)$$

A natural definition for the *initial* state s_0 is

$$s_0 \equiv \mathbf{0} \quad (2.9)$$

i.e., the state in which the system is idle and empty of any process instances. Since the main logical concern in this book is the establishment of deadlock-free execution of all activated process instances, we define S_m as

$$S_m \equiv \{s_0\} \quad (2.10)$$

Hence, the marked language \mathcal{L}_m of this automaton corresponds to "*complete (processing) runs*".

1.3 Example

To exemplify the FSA-based modelling of the RAS behavior, consider a small RAS where the system resource set is $\mathcal{R} = \{R_1, R_2, R_3\}$, with $C_i = 1$, $i = 1, 2, 3$. The process types supported by the system possess the linear structure described by the following resource allocation sequences:

$$\Pi_1 = < \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} >$$

$$\Pi_2 = < \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} >$$

The state of this RAS has six components, corresponding to each of the six processing stages; in particular, state $s_0 = (0 \ 0 \ 0 \ 0 \ 0 \ 0)^T$ denotes the initial empty state. Table 2.1 enumerates the RAS state transition function. As it can be seen in this table, the considered RAS can present 27 distinct allocation states, i.e., $|S| = 27$, with the state signatures running from 0 to 26. Figure 2.1 provides the corresponding *State Transition Diagram (STD)*.

It is interesting to notice that the RAS considered in this example and the derived FSA provide an effective characterization of the untimed / logical dynamics of the small robotic cell considered in the introductory example of Chapter 1. Indeed, the reader can convince herself that the allocation of the robotic manipulator to a process instance is not of particular interest in the logical analysis of the resource allocation taking place in that cell, since this allocation only facilitates the process transfers among the various workstations, and will remain problem-free as long as the allocation of the workstation capacity is properly managed. As a more general principle, developing and exploiting this type of insights in the problem formulation and analysis is very important since it can lead to a pertinent economical representation of the problem under consideration, and thus, it constitutes a first step towards managing its complexity.

◇

2. State Reachability, Safety, and Nonblocking Supervision

In this section, we use the STD of the example RAS presented in Section 1.3, in order to motivate and define some additional concepts that are necessary for the eventual formal definition of the RAS nonblocking SC problem.

Table 2.1. Example: The RAS State Transition Function

s_i	State Vector	Succ. States	s_i	State Vector	Succ. States
0	000000	1, 2	14	000111	8
1	100000	3, 15	15	100100	16, 17
2	000100	4, 15	16	010100	18
3	010000	5, 6, 16	17	100010	19
4	000010	7, 8, 17	18	110100	
5	110000	9, 18	19	100110	
6	001000	0, 9	20	001001	6, 7
7	000001	0, 10	21	010101	16
8	000110	10, 19	22	101010	17
9	101000	1, 11	23	010001	20, 21
10	000101	2, 12	24	001010	20, 22
11	011000	3, 13	25	011001	11, 23
12	000011	4, 14	26	001011	12, 24
13	111000	5			

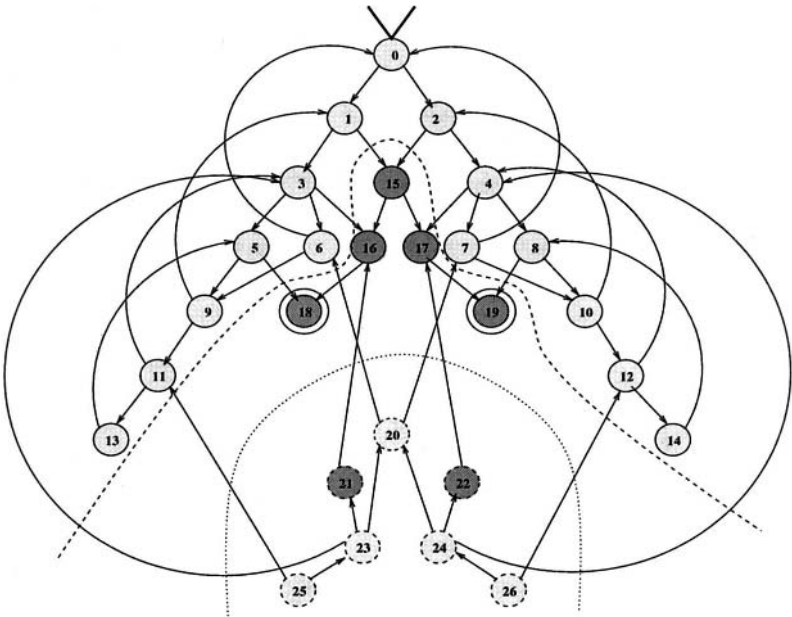


Figure 2.1. Example: The RAS State Transition Diagram

2.1 State Reachability and Safety

As indicated in the introduction of the FSA model, a *directed path* in the STD of Figure 2.1 represents a feasible sequence of events that can take place in the behavior of the considered RAS. We are mainly interested in paths that start and finish at the empty state s_0 . Notice that there is a subset of nodes, shown as dashed nodes in the depicted STD, that cannot be reached from state s_0 through any directed path. This implies that when the system is started from empty state, and operated according to the resource allocation dynamics expressed by Equation 2.7, the resource allocation states 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 they will be called *reachable* states. Formally,

DEFINITION 5 *State s' is reachable from state s , denoted by $s' \leftarrow s$, or equivalently, $s \rightarrow s'$, if and only if (iff) there exists a sequence of events that can bring the system from state s to state s' . In the FSA notation,*

$$\forall s, s' \in S, s' \leftarrow s \iff s \rightarrow s' \iff \exists u \in E^* : f(s, u) = s'$$

Furthermore, a state $s \in S$ will be called a reachable state, iff $s \leftarrow s_0$.

The set of reachable states will be denoted by S_r and the set of unreachable states will be denoted by $S_{\bar{r}}$; obviously $S_{\bar{r}} = S \setminus S_r$. The STD subgraph consisting of the reachable states S_r and the arcs emanating from them is called the *reachability graph* of the FSA.

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 2.1, the former are lightly shaded while the latter are heavily shaded. If the RAS enters any of the heavily shaded states it will never be able, 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*. Formally,

DEFINITION 6 *State s is a safe state iff state s_0 is reachable from state s . A state which is not safe will be called an unsafe state. In the FSA notation,*

$$\forall s \in S, \text{safe}(s) \iff \exists u \in E^* : f(s, u) = s_0 \iff s_0 \leftarrow s$$

The set of safe states is denoted by $S_s \subseteq S$ and the set of unsafe states is denoted by $S_{\bar{s}}$. Again, it holds that $S_{\bar{s}} = S \setminus S_s$. Furthermore, we extend the

characterization of safety to RAS transitions emanating from safe states, by characterizing them as safe if they result in a safe state; mathematically,

$$\forall s \in S_s, \forall e \in \Gamma(s), \text{safe}(e|s) \iff f(s, e) \in S_s$$

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

2.2 Safety and the RAS Deadlock

In the class of DIS-CON-RAS, a *deadlock* state is a state where there exists a subset of activated processes, such that every process in this subset requests resources for its advancement currently held by some other process(-es) in the set. More formally,

DEFINITION 7 *Given a RAS state s , define the apparent slack of resource R_i with respect to (w.r.t.) a set of active processes, \mathcal{DP} , to be equal to the capacity of R_i , C_i , minus the number of its units allocated to processes in \mathcal{DP} .*

Then, state s is a (partial) deadlock, if there exists a set of active processes, \mathcal{DP} , with every process j_j in it requesting a number of units from a resource R_{i_j} that is larger than the apparent slack of R_{i_j} w.r.t. set \mathcal{DP} .

Some deadlock states in the STD of Figure 2.1, are states s_{16} , s_{17} , s_{18} and s_{19} . The class of *deadlock* states for a given RAS configuration will be denoted by S_d .

Deadlocks are the natural reason for the existence of the unsafe states in the RAS operation. This is established by the following two propositions, originally proven in (Reveliotis, 1996):

PROPOSITION 1 *A RAS deadlock state is an unsafe state, i.e., $S_d \subseteq S_{\bar{s}}$.*

Proof: Consider a state s satisfying the deadlock characterization of Definition 7. According to the logic expressed by Equation 2.7, a transition corresponding to a state-event pair is feasible *iff* the corresponding resource request can be met by the system slack capacity. Furthermore, resource units become idle only when an occupying process instance proceeds to its next processing stage or gets unloaded from the system. Neither is possible for the processes in the process subset \mathcal{DP} implied by Definition 7. Thus, all the processes included in \mathcal{DP} cannot proceed to completion, and therefore, the RAS empty set s_0 is unreachable from s . \diamond

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

Proof: Suppose not. Let s_i be an unsafe state on one of the paths emanating from s . Since, by the working hypothesis, s_i is not a deadlock, by taking as \mathcal{DP}^i the entire set of process instances in the system, it follows that there will exist a process instance whose immediate resource requirements are smaller than the current resource slacks, and therefore, it is able to proceed to its next processing stage. Let s_{i+1} denote the resulting state. By invoking the same argument repeatedly, we can keep reducing the total workload in the system by one unit (i.e., one processing stage) at a time. Given the finiteness of the process routes and assuming that no new process instances are loaded into the system, the system is going to be empty after a sufficient number of events. But this contradicts the definition of unsafety and concludes the proof. \diamond

It should be noticed, however, that there can be unsafe states that are not deadlocks. As an example, consider state s_{15} in the STD of Figure 2.1: This state, although one step away from deadlock, it does not contain a deadlock itself, since both of the running process instances can advance to their next requested resource R_2 . These states will be referred to as *deadlock-free unsafe* states and, as it will be shown in the following, they are the main source for the non-polynomial complexity of the maximally permissive nonblocking supervision in the considered RAS class. Before, however, getting into these complexity considerations, we need to formally characterize the concept of (maximally permissive) nonblocking supervisor for the considered RAS class and the corresponding logical control problem.

2.3 Optimal Nonblocking Supervisory Control

The characterization of state safety and its relationship to the RAS deadlock provided in the previous section, facilitates the formal definition of *deadlock avoidance*, which is the deadlock resolution strategy of interest in this book. In the following discussion, it is assumed that the RAS always undergoes normal operation, i.e., its operation is contained in the reachable subspace S_r .

In the FSA modelling context, a *nonblocking supervisory control policy* (SCP) must restrict the operation of a given RAS by limiting it to its reachable and safe subspace S_{rs} . Practically, we seek to identify an appropriate set of feasible transitions which when removed from the STD – or equivalently, *disabled* by the SCP – render the unsafe subspace $S_{r\bar{s}}$ unreachable from state s_0 . At the same time, it must be ensured that every state s 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 to s_0 . States which are reachable under an enforced SCP and from which progress is inhibited by the policy-imposed constraints and not by the RAS structure, are characterized as (*policy*-)*induced* or *restricted* deadlocks in the literature (Banaszak and Krogh, 1990).

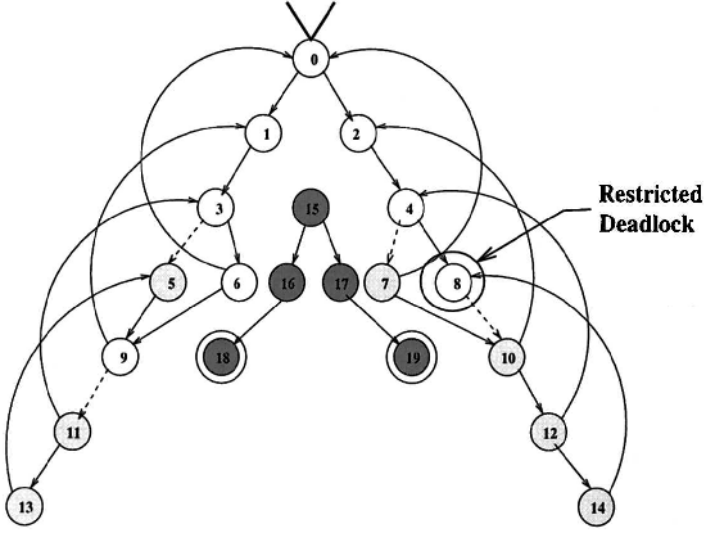


Figure 2.2. Example: An SCP inducing restricted deadlock

An example of an SCP that gives rise to restricted deadlock is presented in Figure 2.2. This hypothetical policy, defined on the RAS-STD of Figure 2.1, *admits*² 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 8 Consider a RAS $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A}, T \rangle$, modelled by the FSA $G = \langle E, S, f, \Gamma, s_0, S_m \rangle$. A (logical) supervisory control policy (SCP) Δ for it is a function

$$\Delta : S \rightarrow 2^E, \text{ with } \Delta(s) = \{e \in \Gamma(s) : e \text{ is selected by the policy}\}$$

Events $e \in \bigcup_{s \in S} \Delta(s)$ are called the policy-enabled or admissible events.

DEFINITION 9 Given an SCP Δ , let $s' \xrightarrow{\Delta} s$ denote the fact that state s' is reachable from state s through an event sequence which comprises policy-enabled events only. Let $S_r(\Delta) = \{s : s \xrightarrow{\Delta} s_0\}$ and $S_s(\Delta) = \{s : s_0 \xrightarrow{\Delta} s\}$. Then, policy Δ is non-blocking or correct iff $S_r(\Delta) \subseteq S_s(\Delta)$.

²i.e., allows access to

In words, an SCP is correct *iff* the policy-reachable subspace $S_r(\Delta)$ 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 *maximally permissive* or *optimal* SCP: A correct SCP Δ^* is *optimal* *iff* the policy restriction on S_{rs} disables *only* those actions that result in unsafe states. Formally,

DEFINITION 10 *A correct SCP Δ^* is optimal iff*

$$\forall s \in S_{rs}, \forall e \in \Gamma(s), e \in \Delta^*(s) \iff f(s, e) \in S_s$$

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

- 1 For a given RAS configuration, the optimal nonblocking SCP Δ^* is *unique*.
- 2 $S_r(\Delta^*) = S_{rs}$. Establishing the optimal control policy Δ^* is equivalent to removing from the reachability graph those transition arcs that belong to the STD *cut* $[S_{rs}, S_{r\bar{s}}]$. For example, in the STD of Figure 2.1, the optimal control policy Δ^* consists of removing the arcs that emanate from lightly shaded solid nodes and cross the twisted dashed line.
- 3 From a computational standpoint, resolving the admissibility of a feasible transition by the optimal SCP Δ^* requires the assessment of the safety of the resulting state; i.e., given a RAS state $s \in S_r$ and a tentative feasible transition corresponding to some labelling event $e \in \Gamma(s)$, the controller implementing the optimal SCP must (i) simulate the execution of transition $f(s, e)$ on the underlying RAS and obtain the resultant state s' , (ii) assess the safety of s' , and (iii) admit the transition *iff* $s' \in S_s$.

The mechanism described in Item 3 above for implementing the optimal SCP belongs to a broader class of SCP's, known as *one-step-lookahead* policies. A formal characterization of this concept is as follows:

DEFINITION 11 *An SCP Δ for a given RAS $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A}, T \rangle$ is characterized as one-step-lookahead, iff there exists a property \mathcal{H} defined on the RAS state space S , such that*

$$\forall s \in S, \forall e \in \Gamma(s), e \in \Delta(s) \iff f(s, e) \in \mathcal{H}(s)$$

The next section establishes that the RAS *safety* decision problem – i.e., "Given s , is $s \in S_s$?" – is NP-complete (Garey and Johnson, 1979) in the considered RAS class, and therefore, the implementation of the optimal SCP will be a computationally intractable problem for most practical applications.

3. The computational complexity of the Optimal Nonblocking SCP

As it was established in the previous section, the computational complexity of the optimal SCP for the considered RAS class is determined by the complexity of the *safety* problem, i.e., given a RAS state s , does there exist a resource allocation sequence that can advance all running processes to completion? In this section we show that, in the general case, this problem belongs to the notorious class of NP-complete problems (Garey and Johnson, 1979). Some results on the computational complexity of the RAS safety problem appeared initially in (Araki et al., 1977; Gold, 1978). The result reported next comes from (Lawley and Reveliotis, 2001), and it can be claimed to be the most general with respect to the RAS classes considered in this book, since it establishes the NP-completeness of safety even for the case of LIN-SU-RAS; we remind the reader that the RAS behavior generated by this class is subsumed by any other class of the RAS taxonomy introduced in Chapter 1.

THEOREM 2.1 *The problem of RAS safety is NP-complete even for RAS in which each process constitutes a linear sequence of processing stages with each stage requesting a single unit from a single resource type for its execution.*

Proof: First of all, notice that safety belongs to NP, since we can verify the validity of any candidate termination sequence, through simulation, in time $O(|\mathcal{R}| \cdot \max_j \{|\mathcal{S}_j|\} \cdot \sum_i C_i)$.

To prove NP-completeness, we provide a polynomial reduction of the 3-SAT problem to the problem of RAS safety, where the latter is restricted in the RAS domain defined in the theorem. A formal statement of the 3-SAT problem is as follows (Garey and Johnson, 1979):

DEFINITION 12 *Let $\chi = \{X_1, \bar{X}_1, X_2, \bar{X}_2, \dots, X_\mu, \bar{X}_\mu\}$ be a set of literals and $\Lambda = \Lambda_1 \wedge \Lambda_2 \wedge \dots \wedge \Lambda_\nu$ be a conjunction of clauses of the form $\Lambda_q = Y_{q1} \vee Y_{q2} \vee Y_{q3}$, where $Y_{qk} \in \chi$, i.e., each Y_{qk} is one of the literals belonging to χ . Let $K \subseteq \chi$ be such that*

CONDITION 4 $\forall i = 1, \dots, \mu$, either $X_i \in K$ or $\bar{X}_i \in K$, but not both.

Then, the satisfiability question is as follows: Given χ and Λ , does there exist $K \subseteq \chi$ satisfying Condition 4 such that $\forall q = 1, \dots, \nu$, $K \cap \Lambda_q \neq \emptyset$?

An instance of the 3-SAT problem, defined by $\langle \chi, \Lambda \rangle$, can be reduced polynomially to the considered safety problem as follows:

For each clause $\Lambda_q \in \Lambda$, define seven processes Π_{q1} to Π_{q7} , with the following resource sequences

$$\Pi_{q1} = \langle \Lambda_{q1}, \Lambda_{q4}, \Lambda_{q5}, \Lambda_{q6}, \Lambda_{q8}, Y_{q1} \rangle, \quad \Pi_{q5} = \langle \Lambda_{q6}, \Lambda_{q8}, \Lambda_{q1}, \Lambda_{q2}, \Lambda_{q3} \rangle$$

$$\begin{aligned}
\Pi_{q2} &= \langle \Lambda_{q2}, \Lambda_{q4}, \Lambda_{q5}, \Lambda_{q6}, \Lambda_{q8}, Y_{q2} \rangle, & \Pi_{q6} &= \langle \Lambda_{q8}, \Lambda_{q1}, \Lambda_{q2}, \Lambda_{q3} \rangle \\
\Pi_{q3} &= \langle \Lambda_{q3}, \Lambda_{q4}, \Lambda_{q5}, \Lambda_{q6}, \Lambda_{q8}, Y_{q3} \rangle, & \Pi_{q7} &= \langle \Psi_q, \Lambda_{q8}, \Lambda_{q6}, \Lambda_{q7} \rangle \\
\Pi_{q4} &= \langle \Lambda_{q7}, \Lambda_{q6}, \Lambda_{q8}, \Lambda_{q1}, \Lambda_{q2}, \Lambda_{q3} \rangle
\end{aligned}$$

Note that for each clause, Λ_q , we define twelve resources $\{\Lambda_{q1} \dots, \Lambda_{q8}, Y_{q1}, Y_{q2}, Y_{q3}, \Psi_q\}$. Furthermore, in agreement with the definition of the 3-SAT problem, each resource Y_{qk} , $k = 1, 2, 3$, represents some X or \bar{X} in χ . Next, for each pair of 3-SAT literals, X_i and \bar{X}_i , define a pair of processes, Π_{i1} and Π_{i2} , with the following resource sequences:

$$\Pi_{i1} = \langle X_i, B_i, D_1, D_2 \rangle, \quad \Pi_{i2} = \langle \bar{X}_i, B_i, D_1, D_2 \rangle$$

Note that for each i , we define resources $\{X_i, \bar{X}_i, B_i\}$. We also define two "global" resources $\{D_1, D_2\}$. Next, define two processes, Π_{01} and Π_{02} , with resource sequences:

$$\Pi_{01} = \langle D_1, \Psi_1, \Psi_2, \dots, \Psi_{\nu-1}, \Psi_\nu \rangle$$

$$\Pi_{02} = \langle D_2, D_1, \Psi_\nu, \Psi_{\nu-1}, \dots, \Psi_2, \Psi_1 \rangle$$

Finally, advance every process to its first stage in the corresponding sequence; the resulting RAS state is depicted in Figure 2.3. Note that the number of processes defined is $2\mu + 7\nu + 2$, the number of resources is $3\mu + 9\nu + 2$, and μ and ν were respectively defined in Definition 12 as the cardinality of set χ and the number of clauses. Therefore, the presented reduction is polynomial.

Now, suppose that Λ is satisfiable. Then there exists K such that $K \cap \Lambda_q \neq \emptyset$, $\forall q = 1, \dots, \nu$. For each $i = 1, \dots, \mu$, if $X_i \in K$, advance Π_{i1} to B_i ; otherwise, advance Π_{i2} to B_i . Figure 2.4 gives the resulting state. Notice that, for each $q = 1, \dots, \nu$, at least one resource in the set $\{Y_{q1}, Y_{q2}, Y_{q3}\}$ is free. Next we show that for each $q = 1, \dots, \nu$, the processes $\Pi_{q1}, \Pi_{q2}, \Pi_{q3}, \Pi_{q4}, \Pi_{q5}, \Pi_{q6}$ and Π_{q7} can be advanced to a point where Ψ_q is released without incurring unsafeness. We consider three cases:

Case 1: $Y_{q1} \in K \cap \Lambda_q$. If Y_{q1} is free, advance processes $\Pi_{q1}, \Pi_{q2}, \Pi_{q3}, \Pi_{q4}, \Pi_{q5}, \Pi_{q6}$ and Π_{q7} as follows: Π_{q1} to Λ_{q4} to Λ_{q5} ; Π_{q2} to Λ_{q4} ; Π_{q6} to Λ_{q1} to Λ_{q2} ; Π_{q5} to Λ_{q8} to Λ_{q1} ; Π_{q1} to Λ_{q6} to Λ_{q8} to Y_{q1} and out of the system; Π_{q2} to Λ_{q5} ; Π_{q3} to Λ_{q4} ; Π_{q6} to Λ_{q3} and out of the system; Π_{q5} to Λ_{q2} to Λ_{q3} and out of the system; Π_{q4} to Λ_{q6} to Λ_{q8} to Λ_{q1} to Λ_{q2} to Λ_{q3} and out of the system; Π_{q7} to Λ_{q8} to Λ_{q6} to Λ_{q7} and out of the system. Figure 2.5(a) shows the resulting state. Note that every Π_q has finished, except for Π_{q2} and Π_{q3} , and that resource Ψ_q is free.

Case 2: $Y_{q2} \in K \cap \Lambda_q$. If Y_{q2} is free, advance processes $\Pi_{q1}, \Pi_{q2}, \Pi_{q3}, \Pi_{q4}, \Pi_{q5}, \Pi_{q6}$ and Π_{q7} as follows: Π_{q2} to Λ_{q4} to Λ_{q5} ; Π_{q1} to Λ_{q4} ; Π_{q6} to Λ_{q1} to Λ_{q2} ; Π_{q5} to Λ_{q8} to Λ_{q1} ; Π_{q2} to Λ_{q6} to Λ_{q8} to Y_{q2} and out of the system; Π_{q1} to Λ_{q5} ; Π_{q3} to Λ_{q4} ; Π_{q6} to Λ_{q3} and out of the system; Π_{q5} to Λ_{q2} to Λ_{q3} and out of the

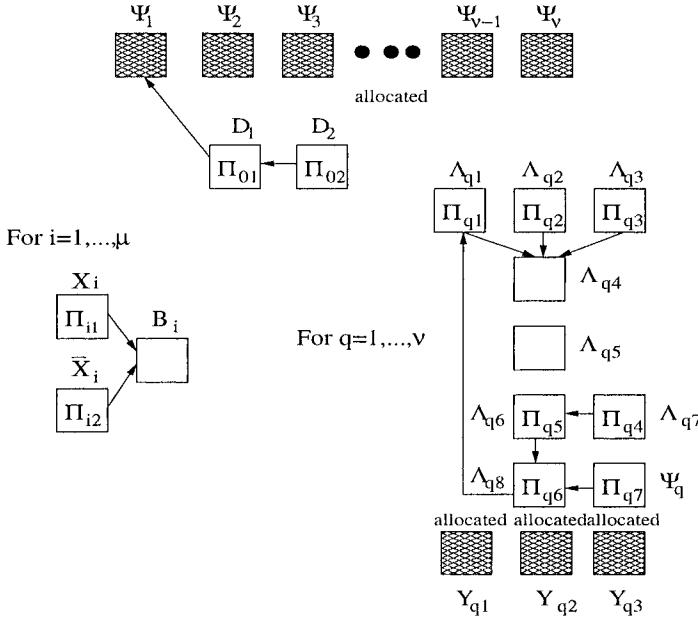
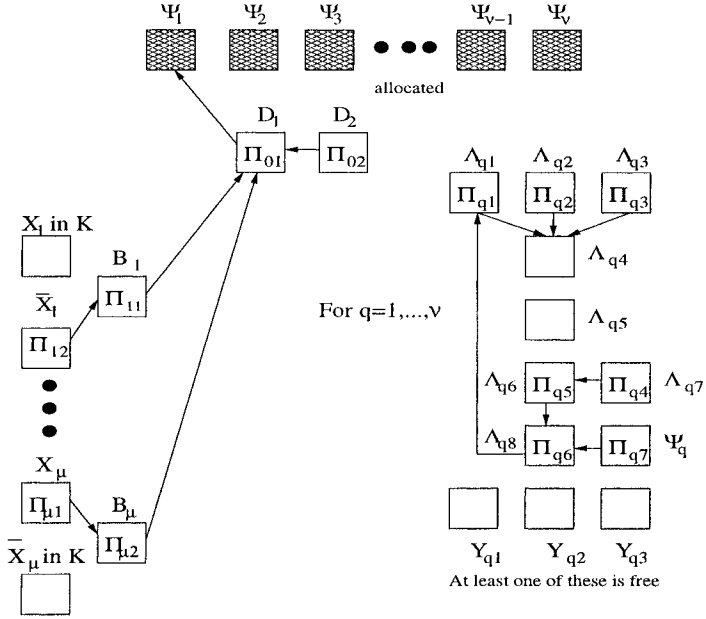
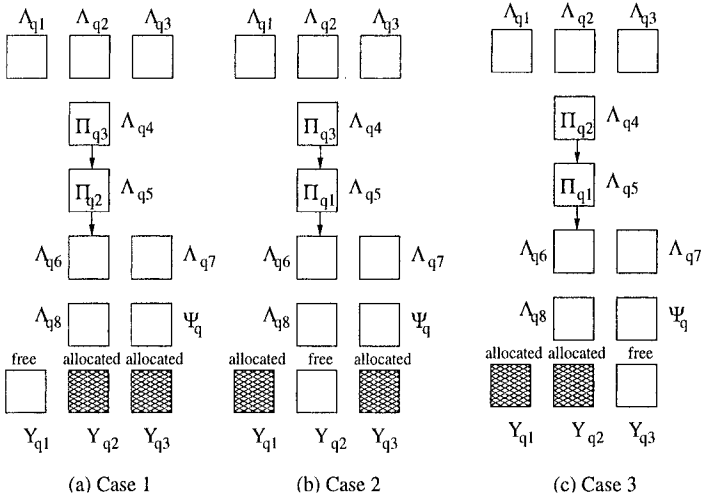


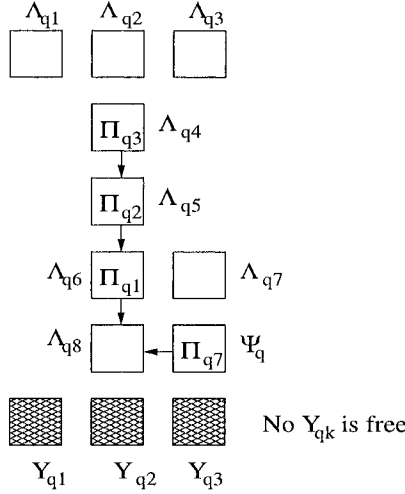
Figure 2.3. The RAS state induced by the 3-SAT problem

system; Π_{q4} to Λ_{q6} to Λ_{q8} to Λ_{q1} to Λ_{q2} to Λ_{q3} and out of the system; Π_{q7} to Λ_{q8} to Λ_{q6} to Λ_{q7} and out of the system. Figure 2.5(b) shows the resulting state. Note that every Π_q has finished, except for Π_{q1} and Π_{q3} , and that resource Ψ_q is free.

Case 3: $Y_{q3} \in K \cap \Lambda_q$. If Y_{q3} is free, advance processes $\Pi_{q1}, \Pi_{q2}, \Pi_{q3}, \Pi_{q4}, \Pi_{q5}, \Pi_{q6}$ and Π_{q7} as follows: Π_{q3} to Λ_{q4} to Λ_{q5} ; Π_{q1} to Λ_{q4} ; Π_{q6} to Λ_{q1} ; Π_{q5} to Λ_{q8} ; Π_{q3} to Λ_{q6} ; Π_{q1} to Λ_{q5} ; Π_{q2} to Λ_{q4} ; Π_{q6} to Λ_{q2} to Λ_{q3} and out of the system; Π_{q5} to Λ_{q1} to Λ_{q2} to Λ_{q3} and out of the system; Π_{q3} to Λ_{q8} to Y_{q3} and out of the system; Π_{q4} to Λ_{q6} to Λ_{q8} to Λ_{q1} to Λ_{q2} to Λ_{q3} and out of the system; Π_{q7} to Λ_{q8} to Λ_{q6} to Λ_{q7} and out of the system. Figure 2.5(c) shows the resulting state. Note that every Π_q has finished, except for Π_{q1} and Π_{q2} , and that resource Ψ_q is free.

Therefore, for $q = 1, \dots, \nu$, it is possible to complete all processes in the set $\{\Pi_{q4}, \Pi_{q5}, \Pi_{q6}, \Pi_{q7}\}$, thus releasing resource Ψ_q . Furthermore, it is possible to complete at least one of Π_{q1}, Π_{q2} or Π_{q3} , with the remaining processes being in one of the states in Figure 2.5. Since Ψ_q is now free, for $q = 1, \dots, \nu$, advance Π_{01} to $\Psi_1, \Psi_2, \dots, \Psi_\nu$ and out of the system. Next, advance Π_{02} to $D_1, \Psi_\nu, \Psi_{\nu-1}, \dots, \Psi_1$ and out of the system. Now, since D_1 and D_2 are free, each Π_{i1} and Π_{i2} can be advanced to D_1, D_2 and out of the system,

Figure 2.4. The RAS state after the release of all resources in K Figure 2.5. A safe deallocation of Ψ_q , $q = 1, \dots, \nu$

Figure 2.6. The case where $\Lambda_q \cap K = \emptyset$

for $i = 1, \dots, \mu$, thus releasing all resources X_i and \bar{X}_i . Therefore, every resource in the set $\{Y_{q1}, Y_{q2}, Y_{q3}\}$ is free for $q = 1, \dots, \nu$. Hence, for every $q = 1, \dots, \nu$, advance all remaining Π_{qk} on Λ_{q5} to $\Lambda_{q6}, \Lambda_{q8}, Y_{qk}$ and out of the system. Then, advance all remaining Π_{qk} on Λ_{q4} to $\Lambda_{q5}, \Lambda_{q6}, \Lambda_{q8}, Y_{qk}$ and out of the system. Since all processes are completed and all resources are released, the state of Figure 2.3 is safe.

Now suppose that Λ is not satisfiable. We shall show that the state of Figure 2.3 is not safe. Since Λ is not satisfiable, for every $K \subseteq \chi$ satisfying Condition 4, there exists Λ_q such that $K \cap \Lambda_q = \emptyset$. Without loss of generality, consider a given K and $i \in \{1, \dots, \mu\}$. If $X_i \in K$, advance Π_{i1} to B_i ; otherwise, advance Π_{i2} to B_i . Next we show that, starting from the state of Figure 2.4, if Π_{q7} releases Ψ_q before a Y_{qk} is available, deadlock results. Note that if Π_{q7} advances to Λ_{q8} (and releases Ψ_q) before Π_{q4} advances to Λ_{q1} , deadlock involving Π_{q7} and Π_{q4} is inevitable. Therefore, we must advance processes so that Π_{q4} can gain Λ_{q1} . Starting from the state given in Figure 2.4 and without loss of generality, advance the processes as follows: Π_{q1} to Λ_{q4} to Λ_{q5} ; Π_{q2} to Λ_{q4} ; Π_{q6} to Λ_{q1} to Λ_{q2} ; Π_{q5} to Λ_{q8} to Λ_{q1} ; Π_{q4} to Λ_{q6} to Λ_{q8} ; Π_{q1} to Λ_{q6} ; Π_{q2} to Λ_{q5} ; Π_{q3} to Λ_{q4} ; Π_{q6} to Λ_{q3} and out of the system; Π_{q5} to Λ_{q2} to Λ_{q3} and out of the system; Π_{q4} to Λ_{q1} to Λ_{q2} to Λ_{q3} and out of the system. These advancements yield the state of Figure 2.6. Note that Π_{q1} cannot proceed beyond Λ_{q8} since none of the Y_{qk} are available. At this point, if Π_{q7} advances to Λ_{q8} and releases Ψ_q , it deadlocks with Π_{q1} . Therefore, if $K \cap \Lambda_q = \emptyset$, Ψ_q must not be released, otherwise, Π_{q7} becomes involved in deadlock.

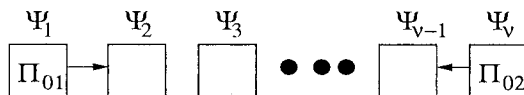


Figure 2.7. The unsafe state involving Π_{01} and Π_{02}

Therefore, in the state of Figure 2.4, there exists at least one set of processes $\{\Pi_{q1}, \Pi_{q2}, \Pi_{q3}, \Pi_{q7}\}$ that cannot be completed until additional Y_{qk} 's are released. Furthermore, the corresponding Ψ_q will not be released. To release additional Y_{qk} 's, we must free some B_i resources by advancing P_{01} . If $K \cap \Lambda_1 = \emptyset$, Ψ_1 is not available. Π_{01} cannot advance, no other Y_{qk} 's can be released, at least one set of processes of the form $\{\Pi_{q1}, \Pi_{q2}, \Pi_{q3}, \Pi_{q7}\}$ cannot be completed, and the system is unsafe. If Ψ_1 is available, advance Π_{01} to Ψ_1 . We must now advance Π_{02} to D_1 , for if we advance any Π_i from B_i to D_1 , it deadlocks with Π_{02} . Our only next choice is to advance Π_{02} to Ψ_ν , thus freeing D_1 . If $K \cap \Lambda_\nu = \emptyset$, then Ψ_ν is not available, P_{02} cannot be advanced, no other Y_{qk} 's can be released, at least one set of processes of the form $\{\Pi_{q1}, \Pi_{q2}, \Pi_{q3}, \Pi_{q7}\}$ cannot be completed, and the system is unsafe. If Ψ_ν is available, advance Π_{02} to Ψ_ν . D_1 and D_2 are now free, so all Π_i processes can be completed, one at a time, and all resources X_i and \bar{X}_i can be released. Therefore, for $q = 1, \dots, \nu$, all process sets $\{\Pi_{q1}, \Pi_{q2}, \Pi_{q3}, \Pi_{q4}, \Pi_{q5}, \Pi_{q6}, \Pi_{q7}\}$ can be completed, and all resources Ψ_q can be released. Only Π_{01} and Π_{02} remain to be completed (see Figure 2.7). Unfortunately, Π_{01} and Π_{02} are headed for an inevitable deadlock. To see this, note that Π_{01} holds Ψ_1 and requires the resource sequence $\langle \Psi_2, \Psi_3, \dots, \Psi_{\nu-1}, \Psi_\nu \rangle$, while Π_{02} holds Ψ_ν and requires the resource sequence $\langle \Psi_{\nu-1}, \Psi_{\nu-2}, \dots, \Psi_2, \Psi_1 \rangle$. Therefore, it is impossible to complete these two processes and the system is unsafe. \diamond

The next theorem establishes that, while RAS safety is an NP-complete problem, the RAS deadlock itself is polynomially recognizable. Therefore, it can be concluded that *the super-polynomial complexity of the RAS safety problem is due to the difficulty of recognizing deadlock-free unsafe states.*

THEOREM 2.2 *In the class of Disjunctive / Conjunctive RAS, the problem of determining whether a given RAS state is a deadlock or not, is of polynomial complexity with respect to the underlying RAS size.*

Proof: An algorithm for resolving this problem is presented in Figure 2.8. This algorithm scans repetitively the non-zero components of the state vector s , setting each of them to 0 upon the identification of a resource allocation request that (i) enables the transition of the corresponding process instances to one of their successor processing stages, and (ii) it can be satisfied from the resource slacks corresponding to the running value of s . If the algorithm succeeds to set

Deadlock Detection Algorithm for DIS-CON-RAS**Input:** DIS-CON-RAS $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A}, T \rangle$ and a state s **Output:** Boolean variable DEADLOCK

```

begin
  /* Initialize */
   $s' := s$ 
  DEADLOCK := FALSE;

  /* processing step */
  while ( $s' \neq 0$  and not(DEADLOCK)) do
    begin
      DEADLOCK := TRUE;
      for ( $q:=1$  to  $\dim(s')$ ) do
        begin
          if ( $s'(q) > 0$ )
            begin
              succ:=< ordered list of indices to successor stages for  $q$  >;
              while (DEADLOCK and succ  $\neq$  NULL) do
                begin
                   $p := \text{head}(\text{succ}); \text{succ} := \text{pop}(\text{succ}, p);$ 
                   $i := 1; \text{DEL} := \text{TRUE};$ 
                  while ( $i \leq \dim(\mathcal{R})$  and DEL) do
                    begin
                       $\text{DEL} := (\delta_i(s') \geq A_{[p]}(i) - A_{[q]}(i));$ 
                       $i := i + 1;$ 
                    endwhile
                  if (DEL)
                    begin
                       $s'(q) := 0;$ 
                      DEADLOCK := FALSE;
                    end
                  endwhile
                end
              endwhile
            end
          endfor
        endwhile
      end
    end
  end

```

Figure 2.8. A polynomial deadlock detection algorithm for the Disjunctive / Conjunctive RAS

all the state components to 0, the considered state s is deadlock-free; otherwise, s is declared to be a deadlock.

Next we show that this algorithm is of polynomial complexity. The algorithm must "delete" – i.e., set equal to 0 – at least one state component in order to proceed to its next scan of the state vector. Therefore, the maximum number of "deletion" tests performed by the algorithm, throughout its entire operation, is $O((\dim(s))^2) = O((\sum_{j=1}^n |\mathcal{S}_j|)^2)$. The number of successor stages that must be examined at each of those "deletion" tests is of order $O(\max_{j=1}^n |\mathcal{S}_j|)$, while the computational cost for assessing the feasibility of the transition from a given stage q to a given stage p is $O(|\mathcal{R}|)$. Therefore, the overall complexity of the algorithm is $O(|\mathcal{R}| \max_{j=1}^n |\mathcal{S}_j| (\sum_{j=1}^n |\mathcal{S}_j|)^2)$. \diamond

4. Practical approaches to the RAS logical control problem

As it has been already pointed out, from a practical standpoint, the result of Theorem 2.1 is a negative result; it establishes that, in the general case, the optimal SCP will be computationally intractable in the operational context of the considered RAS classes. The typical reaction in the face of such a result is either (i) to identify *special structure* for the problem in which the target concept – in our case, the optimal SCP – is polynomially computable, or (ii) to try to generate "good" *approximations* of it that can be obtained with polynomial computational cost. These are the tracks that have been followed by the research community in this case, as well. Special RAS structure that enables the implementation of the optimal SCP in polynomial complexity w.r.t. the RAS size, is discussed in Chapter 3. On the other hand, the notion of a "good" polynomial approximation to the optimal SCP has given rise to the concept of *Polynomial Kernel (PK-) SCP's*. Simply stated, the basic idea behind PK-SCP's is 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 – i.e., polynomially computable – *kernel* among the set of reachable and safe states, and gives the methodology its name.

From an implementational viewpoint, the development of PK-SCP's requires the identification of a property $\mathcal{H}(s)$, $s \in S$, such that: (i) $\mathcal{H}(s_0) = \text{TRUE}$, (ii) $\mathcal{H}(s) \Rightarrow \text{safe}(s), \forall s \in S_r$, and (iii) $\mathcal{H}()$ is *polynomially* testable on the system states. Then, by allowing only transitions to states satisfying \mathcal{H} , through one-step lookahead, it can be ensured that the visited states will be safe. An additional important requirement is that (iv) the resulting SCP is *correct*, i.e., the policy-reachable subspace must be strongly connected (cf. Section 2.3, Definition 9). 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 2.3 *A Polynomial-Kernel SCP is correct iff for every state admitted by the policy, there exists a policy-admissible transition, and this transition does not correspond to a loading event.*

The validity of this theorem 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 marked state s_0 . A more formal statement and proof of this theorem, by means of an algebraic characterization of state safety, can be found in (Reveliotis and Ferreira, 1996). It should be noticed that establishing the policy correctness by means of Theorem 2.3, 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.

A last concern is that the developed PK-SCP's are *efficient*, i.e., they provide considerable flexibility for the system operation, by admitting an extensive part of S_{rs} . The design of efficient PK-SCP's that are appropriate for the various classes of the RAS taxonomy defined in Section 2.2 of Chapter 1, is undertaken in Chapters 4 and 5.

5. Some extensions of the RAS logical control problem

In this section we extend the nonblocking SC problem for DIS-CON-RAS in order to account for additional elements in the underlying system behavior. More specifically, the first of these extensions considers the problem of establishing nonblocking supervision when the behavior of the underlying RAS is *uncontrollable* with respect to certain resource allocations. The second extension deals with the problem of *(re-)configuration management*, i.e., the preservation of nonblocking operation in the face of changes in the system structure, e.g., due to resource failures. Since both of these extended versions subsume the basic problem definition considered in the earlier sections of this chapter, it can be concluded that the development of maximally permissive SC policies for them will be computationally intractable. Indeed, the problem of synthesizing effective and computationally efficient SC policies for these extended RAS classes is a challenging problem currently under active investigation. Here, we provide detailed characterizations of the aforementioned problems by means of the FSA modelling framework; some additional available results on them are reported in later chapters.

5.1 Nonblocking supervision under uncontrollable resource allocations

In the entire development of this chapter up to this point, we have implicitly assumed that (i) it is upon the exclusive discretion of the RAS controller to

disable and/or postpone the execution of the various feasible events during the system operation, and that (ii) process instances with flexible routing can be forced, if necessary, to a certain routing option. Here we seek to relax these two assumptions. In order to motivate this need, we notice that, in certain cases, the system might not have the necessary hardware that would facilitate the disabling of certain allocation events. In other cases, the inability of the controller to postpone the execution of certain resource allocations, once they are physically possible, can result from the time-criticality of the involved operations; as a more concrete example for this case, taken from the manufacturing domain, consider a forging operation the must follow the preheating of the part at a certain temperature. Both of these cases refute the first assumption mentioned above, regarding the controllability of the execution of the various feasible events, and they will be collectively characterized as *Type-1 uncontrollability*. The second assumption, regarding the controllability of the process routing, will be naturally violated by systems with non-perfect yield: in these contexts, the eventual routing of each process instance will be contingent upon the outcome of its current processing stage, and it will not be (exclusively) determined by the controller. We shall refer to this type of uncontrollability, resulting from uncontrollable routing, as *Type-2 uncontrollability*.

Our intention is to develop an SCP that will maintain nonblocking operation for the underlying RAS in spite of the occurrence of Type-1 and/or Type-2 uncontrollable events. We start by providing the formal specifications for such a supervisor in the FSA-modelling context. To facilitate the subsequent discussion, let E_u^1 and E_u^2 respectively denote the Type-1 and Type-2 uncontrollable events of the considered RAS; $E \setminus (E_u^1 \cup E_u^2) \equiv E_c$ will be the set of *controllable* events.

DEFINITION 13 *An SCP Δ is acceptable w.r.t. E_u^1 iff $\forall s \in S, (e \in \Gamma(s)) \wedge (e \in E_u^1) \implies e \in \Delta(s)$.*

DEFINITION 14 *An SCP Δ is acceptable w.r.t. E_u^2 iff $\forall s \in S, (e \in \Gamma(s)) \wedge (e \in E_u^2) \implies \exists$ a state sequence $s \xrightarrow{\Delta(s)} s_1 \xrightarrow{\Delta(s_1)} s_2 \xrightarrow{\Delta(s_2)} \dots \xrightarrow{\Delta(s_{n-1})} s_n$ s.t.*

1 $e \in \Delta(s_n)$;

2 if e corresponds to the advancement of stage Ξ_{jk} , then, $\forall \nu = 1, \dots, n$, $s_\nu(q(j, k)) \geq s(q(j, k))$.

In words, an acceptable SCP Δ cannot delay the occurrence of Type-1 uncontrollable events, and it cannot prevent the occurrence of Type-2 uncontrollable events, even though it might postpone the latter until some other events have occurred. Under the reasonable assumptions that (i) all loading events are controllable, and (ii) resource allocation vectors are well-defined w.r.t. the resource

Input: FSA $G = \langle E_c \cup E_u^1 \cup E_u^2, S, f, \Gamma, s_0, \{s_0\} \rangle$ representing a DIS-CON-RAS

Output: Maximally permissive acceptable nonblocking SCP Δ^* and the policy-admissible space $S_r(\Delta^*)$

- 1 Generate the reachability space S_r for the considered RAS.
- 2 Extract the reachable and safe subspace S_{rs} . One effective way to execute this operation is by reversing the arcs in the STD corresponding to S_r , and marking all states that can be reached from s_0 in the modified STD; the marked nodes correspond to S_{rs} .
- 3 Consider the policy Δ that restricts the system operation in S_{rs} , and compute the set $S_D \subseteq S_{rs}$ for which Δ fails to satisfy the acceptance condition of Definition 13.
- 4 If $S_D = \emptyset$, proceed to Step 5; otherwise, set $S_r := S_{rs} \setminus S_D$, and go to Step 2.
- 5 Consider the policy Δ that restricts the system operation in S_{rs} , and compute the set $S_D \subseteq S_{rs}$ for which Δ fails to satisfy the acceptance condition of Definition 14.
- 6 If $S_D = \emptyset$, proceed to Step 7; otherwise, set $S_r := S_{rs} \setminus S_D$, and go to Step 2.
- 7 Return $\Delta^* := \Delta$ and $S_r(\Delta^*) := S_{rs}$.

Figure 2.9. An algorithm computing the maximally permissive acceptable nonblocking SCP for DIS-CON-RAS with Type-1 and/or Type-2 uncontrollability

capacities – i.e., $A_{jk}(i) \leq C_i, \forall i, j, k$ – such a policy will always exist: this is the policy that allows only one process instance in the system. Next, we characterize also the *maximally permissive* nonblocking supervisor Δ^* for this extended RAS class. In particular, Figure 2.9 presents an algorithm that, when executed on any given RAS belonging to the considered sub-class, will return the maximally permissive nonblocking acceptable SCP Δ^* .

The algorithm of Figure 2.9 starts with the reachable and safe space S_{rs} , that characterizes the system behavior under the maximally permissive SCP, when the system is totally controllable, and subsequently, it removes all states that fail to satisfy the acceptance condition of Definition 13 w.r.t. the policy Δ induced by S_{rs} . These are states from which the system can transition uncontrollably to its unsafe region, and therefore, they must be rendered inaccessible by any acceptable policy. However, the removal of these states from S_{rs} will lead to

a reduced space S_r that might not be strongly connected and, as a result, the policy Δ induced by it, will not be correct. Hence, the trimming operation of Step 2 must be repeated on the reduced S_r . The resulting S_{rs} might still contain states from which the system can transition uncontrollably to $S \setminus S_{rs}$, and therefore, the loop of Steps 2 and 3 is repeated until the identification of a subspace S_{rs} that is closed under Type-1 uncontrollability. At this point the satisfaction of the requirement of Definition 14 is taken on. If it happens that the subspace S_{rs} developed in the last execution of Step 2 satisfies also the requirement of Definition 14, then there is nothing more to be done; if, however, there are policy-admissible states violating this new condition, then they must be eliminated, and the algorithm returns to Step 2, in order to generate a more restrictive policy that satisfies the correctness and acceptance requirements of Definitions 9, 13 and 14. A more formal proof for the correctness of this algorithm can be obtained through the linguistic frameworks of (Cassandras and Lafortune, 1999; Kumar and Garg, 1995).

The computation involved in the above algorithm also establishes that Δ^* is uniquely defined. However, as it was indicated in the opening discussion of this section, the result of Theorem 2.1 implies that the implementation of Δ^* is an *NP*-hard problem (Garey and Johnson, 1979). A methodology that generates polynomial, correct and acceptable, sub-optimal SCP's for DIS-CON-RAS with Type-1 and/or Type-2 uncontrollability, is presented in Section 5 of Chapter 5.

5.2 Maintaining nonblocking operation under resource failures

All the above analysis has presumed a stable system configuration; in particular, it has been assumed that the resource capacities and the set of job types supported by the system does not change over time. In this section we consider how to effectively accommodate in the applied control logic any changes in these two elements of the system configuration. From a conceptual standpoint, there are two main approaches that can be adopted with respect to this issue:

The reactive approach As suggested by its name, this approach simply reacts to the various contingencies taking place in the system, by developing a new control policy for the emerging RAS configuration. Hence, in the context of this approach, it is important that (i) the applied SCP's can be algorithmically developed for any given RAS configuration Φ , and furthermore, (ii) the complexity of the algorithms involved in the policy development is fairly low. An additional consideration is to design the new policy in a way that it can accommodate the running RAS state with minimal disruption, i.e., the number – or, more generally, a cost measure determined by the set – of active processes that must be rearranged or unloaded from the system, in order to obtain a RAS state admissible by the new policy, must

be minimized. In general, this approach is more appropriate for systems in which the aforementioned disruptions, and the associated need for policy reconfiguration, are rather rare.

The proactive approach Under this approach, there is one single policy that controls the system operation under all emerging configurations. The applied control logic anticipates potential disruptions in the system operation, and it seeks to ensure that during the occurrence of any such disruption, the system operation will degrade in a "*graceful*" manner. In the particular case of a resource failure, the applied policy logic seeks to ensure that the system will still be able to maintain the execution of all those process types that are not requesting the failing resource(s) for their execution. This is to be achieved without any external intervention, but simply by exploiting those resources whose operation is necessarily stalled because of the experienced failure(s), as a buffer that will accommodate all the process instances blocked in the system because of their need to use the failing resources. A major challenge for this approach is to distribute these blocked jobs in the aforementioned buffering resources in a way that the resulting RAS state will also be policy-admissible in the original RAS configuration, that will be re-established by the repair of the failing resources. The proactive approach seems to be more appropriate for systems experiencing frequent outages, and therefore, the cost / effort of reconfiguring the RAS state and the applied SCP would be too high. On the other hand, by accounting for the "worst-case" scenario at any point in time, SCP's following the proactive approach are quite restrictive / conservative in terms of their permissiveness. From a methodological standpoint, the proactive approach is an application of *robust control* on the RAS supervisory control problem.

Some currently available results with respect to both of these approaches are discussed in Chapter 4.

6. Historical and bibliographical notes

As it was mentioned in the bibliographical discussion of Chapter 1, the problem of deadlock avoidance in sequential resource allocation systems has received particular attention in the last decade, primarily because of its emergence as an important problem in the operation of flexibly automated production systems. Some seminal works that introduced the problem in the academic community are those appearing in (Viswanadham et al., 1990; Banaszak and Krogh, 1990; Wysk et al., 1991). All these papers deal with the problem of deadlock avoidance as it arises in LIN-SU-RAS, since their primary motivation is the effective allocation of the buffering capacity in contemporary flexibly automated manufacturing systems. From a methodological standpoint, the work of (Viswanadham et al., 1990) establishes the ability of the FSA modelling

framework to provide a systematic characterization of the deadlock avoidance problem. The authors recognize the very high complexity of the problem – although they do not provide any formal results for it – and suggest to deal with this complexity through partial lookahead; their approach tends to reduce the occurrence of deadlock but it does not eliminate it completely. The first well-known provably correct deadlock avoidance policy of polynomial complexity for LIN-SU-RAS was presented in (Banaszak and Krogh, 1990);³ we shall discuss this policy in detail in Chapter 4. The work of (Wysk et al., 1991) investigates more extensively the structure of the circular dependencies characterizing the deadlock in LIN-SU-RAS and seeks to develop a formal representation and an algorithmic approach for their detection; however, the proposed approach presents non-polynomial computational complexity w.r.t. the RAS size, and therefore, it is not scalable.

The appearance of the aforementioned works was followed by a large number of publications seeking to provide a more profound understanding of the deadlock problem and its complexity, as it appears primarily in the SU-RAS context. Some indicative examples of those works can be found in (Wysk et al., 1994; Hsieh and Chang, 1994; Ezpeleta et al., 1995; Barkaoui and Ben Abdallah, 1995; Reveliotis and Ferreira, 1996; Fanti et al., 1997; Lawley et al., 1997b; Lawley et al., 1997c; Lawley et al., 1998a; Reveliotis et al., 1997; Lawley et al., 1998b; Lawley, 1999; Park and Reveliotis, 2000; Lawley and Reveliotis, 2001). More recently, the research community has effectively addressed the deadlock problem arising in more complex RAS structures; indicative results in this direction are the works presented in (Barkaoui et al., 1997; Tricas et al., 1998; Tricas et al., 1999; Park and Reveliotis, 2001b; Park, 2000; Park and Reveliotis, 2002a; Jeng and Xie, 2001; Lawley and Sulistyono, 2002; Jeng et al., 2002; Reveliotis, 2003a). We shall return to many of these works in the subsequent chapters of this book.

Regarding the material developed in this chapter, the characterization of the DIS-CON-RAS behavior and of the corresponding SC problem through the FSA modelling framework, presented in Sections 1 and 2, is based on (Reveliotis, 1996; Reveliotis and Ferreira, 1996). The complexity result of Theorem 2.1 was published in (Lawley and Reveliotis, 2001). To the best of our knowledge, the algorithm employed in the proof of Theorem 2.2 has not appeared anywhere else, although similar ideas have appeared in other works, e.g., (Peterson, 1981; Reveliotis et al., 1997). The idea of PK-SCP's was originated in (Banaszak and Krogh, 1990) and it was further formalized in (Reveliotis, 1996; Reveliotis and Ferreira, 1996). The concepts of RAS uncontrollability discussed in Section 5.1 have originally appeared in (Park, 2000; Park and Rev-

³It must be mentioned, however, that many of the ideas underlying the developments of (Banaszak and Krogh, 1990), originally appeared in (Banaszak and Roszkowska, 1988).

eliotis, 2002a). However, the detailed statement of the acceptable nonblocking SC problem in the FSA-based RAS modelling framework, and the algorithm of Figure 2.9 for the computation of the maximally permissive acceptable nonblocking supervisor, is new material. The concepts and ideas regarding the reactive and proactive approaches to RAS re-configuration management are coming respectively from (Reveliotis, 1999) and (Lawley and Sulistyono, 2002).

From a more general standpoint, the FSA-based formulation of the RAS nonblocking SC problem, provided in Sections 1 and 2, is a straightforward implementation of *Ramadge & Wonham's* SC framework (Ramadge and Wonham, 1989) in the RAS operational context. In particular, the work of (Li and Wonham, 1988) has addressed the more abstract problem of developing maximally permissive nonblocking acceptable supervisors for FSA's with uncontrollably enabled events (corresponding to Type-1 uncontrollability in the modelling framework of Section 5.1). The reader is referred to (Cassandras and Lafortune, 1999) for a systematic introduction to Ramadge & Wonham's SC theory.



<http://www.springer.com/978-0-387-23960-6>

Real-Time Management of Resource Allocation Systems

A Discrete Event Systems Approach

Reveliotis, S.

2005, X, 244 p., Hardcover

ISBN: 978-0-387-23960-6