International Journal of Control
Vol. 81, No. 9, September 2008, 1364–1381

Taylor & Francis
Taylor & Francis Group

Incremental hierarchical construction of modular supervisors for discrete-event systems

R.C. Hill* and D.M. Tilbury

University of Michigan, USA

(*Received 19 April 2007; final version received 7 November 2007*)

Adoption of the supervisory control framework introduced by Ramadge and Wonham has been slowed somewhat by the problem of state space explosion that arises in systems of the scale common to most industrial applications. Hierarchical and modular approaches have been explored historically as means for addressing this problem. The limitations of these approaches include that a 'consistent' hierarchy is often difficult to achieve, and that modular supervisors often conflict with each other when acting in conjunction. This paper offers an approach that addresses some of these issues by incrementally building modular supervisors that are nonconflicting by construction. Abstractions are employed to make the procedure more computationally feasible. Proof is given showing the set of modular supervisors generated in this manner meet given specifications without blocking. Furthermore, examples are provided that demonstrate the reduction in complexity that this approach provides.

Keywords: modular supervisory control; hierarchy; abstraction; observers

1. Introduction

In recent years, a well-formed theory for the control of discrete-event systems (DES) has developed following the supervisory control framework introduced by Ramadge and Wonham (1989). A significant hurdle to the adoption of these methods is the state space explosion that occurs in modelling systems of the size most commonly found in industry.

Consider the Flexible Manufacturing System (FMS) example shown in Figure 1 consisting of six machines and four buffers (de Queiroz et al. 2005). The traditional approach to this control problem is to build a single monolithic supervisor to control the entire combined system. Assuming that each machine model has five states and each buffer model has two states, the maximum possible size of the supervisor for this simple system grows quickly to 250 000 states ($5^6 \times 2^4$).

One possible solution is to apply the methods of *hierarchical supervisory control* (Zhong and Wonham 1990; Wong and Wonham 1996), where the supervisor is designed for an abstracted version of the system. The idea is that by abstracting away details of the system, less computation is necessary if the design or analysis can be performed on the simplified version of the system. The primary limitation here is that maintaining 'consistency' between the original and abstracted models is often difficult and limits either the amount of abstraction, or the quality of results that can be achieved. Nonblocking and optimality of the control are specifically difficult to attain. Another problem is that often the full monolithic system must be built before the abstraction is performed.

Another approach is to build a series of smaller component supervisors that each satisfy only a portion of the global specification, rather than a single monolithic supervisor that satisfies the entire specification all at once. This approach offers significant gains in computational complexity and understandability and is generally referred to as *modular supervisory control* (Ramadge and Wonham 1988; de Queiroz and Cury 2000; Gaudin and Marchand 2004; Komenda et al. 2005). For the FMS example, a modular supervisor would be responsible for monitoring only a single buffer. The dashed boxes of Figure 1 illustrate how the FMS example would be partitioned into closed-loop modules according to the approach of

*Corresponding author. Email: rchill@umich.edu

ISSN 0020–7179 print/ISSN 1366–5820 online

DOI: 10.1080/00207170701799365
http://www.informaworld.com

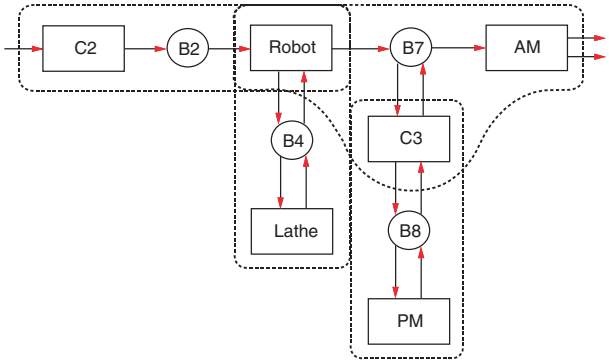

Figure 1. Flexible Manufacturing System example, partition 1.

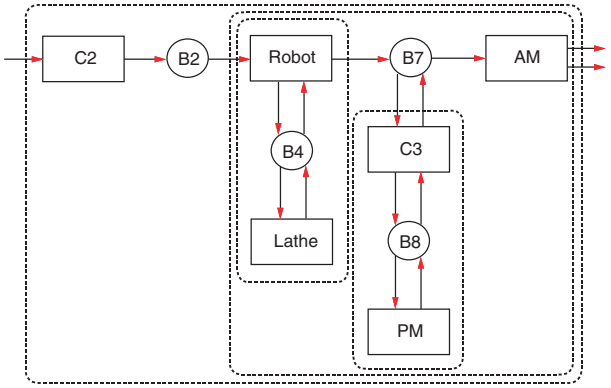

Figure 2. Flexible Manufacturing System example, partition 2.

de Queiroz and Cury (2000). Another methodology referred to as *decentralised supervisory control* builds each component supervisor with respect to an abstracted version of the monolithic plant (Lin and Wonham 1988).

While these modular and decentralised approaches to supervision have been shown to guarantee safety of the controlled system, that is, to maintain system behaviour within the boundaries laid out by the given specifications, there is no guarantee that the system will be able to complete its desired tasks. In other words, the system may block. This is not surprising since the various specifications may have competing goals. To avoid this blocking, it is necessary to verify that the individual closed-loop modules have the property that they are nonconflicting. Unfortunately, the process of verifying nonconflict is approximately as computationally intensive as building the monolithic supervisor in the first place (Ramadge and Wonham 1988). Work has been done to reduce the complexity of verifying nonconflict (Pena et al. 2006; Flordal and Malik 2006a). These works are very useful, but still leave the problem of what to do if conflict is detected. Two special cases that guarantee nonconflict among a set of languages are when the component languages are disjoint and when the languages are subsets of one another.

The approach we propose in this paper addresses the problem of complexity in supervisory control by adopting some of the elements of existing hierarchical and modular approaches, while at the same time avoiding some of their weaknesses. The overall goal is to generate a set of modular supervisors that, when acting in conjunction, result in a controlled system that is guaranteed to meet given specifications and to be nonblocking, without having to build the full unabstracted system and without having to verify that the component supervisors are nonconflicting. The results presented here expand on the authors' previously published work (Hill and Tilbury 2006) and assume the global plant and specification are given modularly. It is further assumed that the plant components are disjoint and interact only through the given specifications, that is, the uncontrolled plant is a *product system*.

Our approach achieves nonconflict by construction by partitioning the system so that the languages representing the closed-loop behaviour of each of the modules satisfy the special properties mentioned earlier, that is, the languages are either disjoint or contained in one another. For the FMS example of Figure 1, a possible partition generated by our proposed approach is shown in Figure 2. The systems in each of the two inner dashed blocks are supervised by their own supervisor and the two outer dashed blocks represent the systems supervised by the third and fourth supervisors. Nonconflict of each of the closed-loop languages is guaranteed since the two modules on the first level of the hierarchy are disjoint, the behaviour allowed by the supervisor on the second level is a subset of the behaviour allowed by the supervisors on the first level and the behaviour allowed by the fourth supervisor is subsequently a subset of the behaviour allowed by any of the preceding supervisors. The problem one might recognise is that the fourth supervisor acts on the full system, which is counter to the goal of a modular approach. The solution is to perform an abstraction on the supervised systems in the inner blocks, before moving up a level of the hierarchy to design the remaining supervisors. If an aspect of a lower level component is not relevant to any of the remaining modular specifications, then it can be abstracted away. In this way, abstractions are performed incrementally, rather than on the entire system at once. Also, a greater level of abstraction can be achieved than if the abstraction were performed on the monolithic system at the very end. For instance, even though the highest level supervisor is built with

respect to the full plant, a significant amount of reduction over the abstracted monolithic solution is generally achievable. This is because the last modular supervisor is only trying to guarantee the additional satisfaction of the last specification, rather than all of the specifications simultaneously.

Some works which similarly aim to combine aspects of hierarchical and decentralised control exist (Lee and Wong 1997; Schmidt et al. 2005). The requirements on the structure of the component supervisors in these works are different than those required here, and are not satisfied by construction. The work of Gaudin and Marchand (2005) presents results that apply specifically to the state avoidance problem. Their approach is able to generate a global supervisor by solving several smaller local state avoidance problems. Research presented in Flordal and Malik (2006b) employs a different type of abstraction based on *supervision equivalence* to incrementally construct the monolithic supervisor for a system. This work is not a modular approach to control, but succeeds in producing a very compact representation of the monolithic solution. It is not completely clear how much, if any, reduction in computational complexity this approach provides.

The work of Wong and Wonham (1998) also applies notions of hierarchical and modular control to reduce complexity. Their approach specifically builds modular supervisors then adds another level of control to resolve the conflict between the supervisors. Abstraction is employed to further reduce the complexity. Nonblocking control is achieved by requiring an observer property of the abstraction and optimality is achieved by additionally requiring an output-control-consistency property. Although the work provides results for general abstractions and control structures and inspires some of the ideas of several subsequent papers (including this one), the work is rather theoretically complex and computational tools do not exist for carrying out the proposed methods. The work of Feng and Wonham (2006a) employs a similar architecture, but considers the specific case of natural projection with the observer property and a supervisory control structure to overcome the theoretical complexity.

Although the methodology of this paper also employs natural projections with the observer property, no conflict-resolving coordinators are built. Rather, each modular supervisor enforces the associated specification as well as nonconflict at the same time. More importantly, the work of this paper sacrifices optimality in order to achieve a greater reduction in complexity. For example, consider the partitioning of modular supervisors shown in Figure 1. In the approach of Feng and Wonham (2006a), a conflict resolving coordinator would be built on the basis of of the abstractions of the modular supervisors. In this process, all events relevant to the components *Robot* and *C*3 would be retained throughout the process since they are shared between the modular supervisors. With the approach of this paper however, at each stage of the process only those events of *Robot* and *C*3 that are relevant to one of the remaining specifications would have to be retained. Additionally, the work of this paper does not require an output-control-consistency property. Specific conditions under which the work of this paper provides optimal control will be left as an open problem.

Other approaches exist (Endsley and Tilbury 2004; Leduc et al. 2005) which also employ another level of control to coordinate the interaction of various components of a large system. These works do not provide explicit algorithms for the construction of their interfaces, but rather depend on the designer's understanding of the system. Furthermore, the work of Endsley and Tilbury (2004) does not address blocking and the work of Leduc et al. (2005) can still suffer from exponential growth, though on a smaller scale.

The outline of this paper is as follows: §2 introduces notation and some definitions; §3 uses a small example to demonstrate the procedure for generating the modular supervisors; §4 demonstrates that the conjunction of these supervisors will provide nonblocking behaviour that meets the given specifications; §5 presents two moderately large examples along with a discussion of the complexity of our approach and §6 summarises the work of the paper and outlines some areas of further investigation.

2. Notation and preliminaries

In this work we will consider DES modeled by automata in the context of the supervisory control framework put forth by Ramadge and Wonham (1989). It is assumed the reader is familiar with the basic notions of supervisory control (Cassandras and Lafortune 1999).

We will in general perform all procedures and proofs employing languages which represent the behaviour of our system. It is however understood that each language is generated and marked by an underlying automaton. For example, an automaton G generates the language $\mathcal{L}(G)$ and marks the language $\mathcal{L}_m(G)$ where marked strings represent a sequence of events that end in a completion state having been reached. The notation $\overline{L}$ represents the set of all prefixes of strings in the language L, and is referred to as the *prefix-closure* of L. An automaton is said to be nonblocking when $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$. For a deterministic

automaton G, this intuitively means that all generated strings can successfully reach a marked state. If a system enters a state from which it cannot reach a marked state, the system is said to have *blocked*.

For the purposes of control, the event set of an automaton is partitioned into *controllable* and *uncontrollable* events, $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$, where controllable events can be disabled and uncontrollable events cannot. Given a set of allowed behaviours $K \subseteq \mathcal{L}_m(G)$ and a set of uncontrollable events $\Sigma_u \subseteq \Sigma$, the existence of a supervisor that can successfully restrict the operation of the plant within the behaviour allowed by the specification is guaranteed by satisfaction of the following Σ_u-*controllability condition* (Cassandras and Lafortune, 1999):

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}. \tag{1}$$

If the above expression holds, it is said that the language K is Σ_u-controllable with respect to the language $\mathcal{L}(G)$. Note, Σ_u-controllability is fundamentally a property of a language's prefix closure.

The operation of two automata together is captured via the synchronous composition (parallel composition) operator, $\|$. In order to precisely define the *synchronous composition* operator, the projection, $P_i : \Sigma^* \to \Sigma_i^*$, is defined as follows:

$$\begin{aligned} P_i(\varepsilon) &:= \varepsilon \\ P_i(e) &:= \begin{cases} e, & e \in \Sigma_i \subseteq \Sigma \\ \varepsilon, & e \notin \Sigma_i \subseteq \Sigma \end{cases} \\ P_i(se) &:= P_i(s)P_i(e), s \in \Sigma^*, e \in \Sigma. \end{aligned} \tag{2}$$

Given a string $s \in \Sigma^*$, the *projection* P_i erases those events in the string that are in the alphabet Σ but not in the subset alphabet Σ_i. We can also define the inverse projection as follows:

$$P_i^{-1}(t) := \{s \in \Sigma^* : P_i(s) = t\}. \tag{3}$$

The effect of the inverse projection P_i^{-1} is to extend the local alphabet Σ_i to Σ. In terms of automata, this is represented by adding self-loops at every state for each event in the set $(\Sigma - \Sigma_i)$. These self-looped events are in essence enabled at every state and as such do not meaningfully restrict the behaviour of the system. With this in mind, we will refer to these events as being irrelevant and will not count them when talking about the total number of transitions in an automaton and will not draw them in figures. We can also define irrelevant events in terms of languages. The following technical definition is taken from Brandin et al. (2004).

Definition 1: For a language $L \subseteq \Sigma^*$, an event $\sigma \in \Sigma$ is said to be *irrelevant* for L, if we have for all $s, t \in \Sigma^*$

$$st \in L \quad \text{if and only if } s\sigma t \in L.$$

Otherwise σ is called relevant for L.

We will use the notation rel(L) to represent the subset of *relevant* events in the event set of the language L.

The projection definitions given by (2) and (3) can be naturally extended from strings to languages and then applied to give a formal definition of the synchronous composition. In the following, $P_i : \Sigma^* \to \Sigma_i^*$ where $\Sigma = \cup \Sigma_i$.

$$L_1 \| L_2 \| \cdots \| L_n := \bigcap_{i=1}^{n} P_i^{-1}(L_i). \tag{4}$$

In addition to determining the existence of a supervisor that can achieve a given specification, it is also desirable that the controlled system be nonblocking. If and only if K is $\mathcal{L}_m(G)$-closed ($K = \overline{K} \cap \mathcal{L}_m(G)$) and Σ_u-controllable with respect to the language L, then a supervisor exists such that the supervised behaviour exactly equals the admissible language $\overline{K}$, and the set of marked behaviours exactly equals K (Cassandras and Lafortune 1999).

In the case that the Σ_u-controllability condition of (1) does not hold, and thus a supervisor able to provide the behaviour of K does not exist, it is desirable to find the largest sublanguage of K for which such a supervisor does exist. This supremal controllable sublanguage is denoted, $\hat{K} = \sup\mathcal{C}(K, L) \subseteq K$. This is thought of as the optimal solution of the supervisory control problem in the sense that it is least restrictive.

In the work of this paper it will be assumed that the plant and specification are given in terms of languages in the following component-wise manner:

$$L_m = L_{m,1} \| \cdots \| L_{m,n}, \text{ and } \overline{K_{\text{spec}}} = \overline{K_{\text{spec},1}} \| \cdots \| \overline{K_{\text{spec},p}}. \tag{5}$$

It will be further assumed that each component has the same event set, though not necessarily the same set of relevant events. Furthermore, it will be assumed that each of the component plants $L_{m,i}$ have been organised such that they do not share relevant events. This situation is referred to as a product system (Ramadge and Wonham 1989).

As stated earlier, modular supervisors will not block one another if they are nonconflicting. Two languages K_1 and K_2 are said to be nonconflicting if they satisfy the relation $\overline{K_1} \cap \overline{K_2} = \overline{K_1 \cap K_2}$. In words, to be nonconflicting means that if K_1 and K_2 share a prefix, they must share a string containing that prefix. In general, determining whether two languages are nonconflicting is a computationally expensive procedure. Two special cases under which languages are guaranteed to be nonconflicting are when languages have disjoint relevant event sets ($\text{rel}(K_1) \cap \text{rel}(K_2) = \emptyset$), and when one language is a

subset of the other ($K_1 \subseteq K_2$) (Cassandras and Lafortune 1999).

As part of our procedure for constructing modular supervisors, an abstraction will be performed. The abstraction that will be employed in this paper is the natural projection defined by (2). The idea here is that if an event is not relevant to any of the remaining specifications, then we do not necessarily need to keep track of it and hence it can be considered for erasure from our models. However, it is possible that erasure of some of these events can hide some aspect of the behaviour of the system that is relevant to the remaining specifications.

Since the growth of the state space comes as a result of the composition of the modules, we would like to apply our abstraction incrementally on each module before a composition is performed. A result that will be helpful to us in this regard is that the projection distributes across the synchronous composition operation if no events relevant to more than one component are erased. This fact is expressed by the following property from Su (2004), where I represents a set of indices:

Proposition 1: *For $i \in I$ let $L_i \subseteq \Sigma^*$ and $L := \bigcap_{j\in I} L_j$. Let $\Sigma_{\text{com}} = \cup\{rel(L_i) \cap rel(L_j) \mid i, j \in I \ \wedge \ i \neq j\}$. Then*

$$\Sigma_{\text{com}} \subseteq \Sigma_k \Rightarrow P_k(L) = \bigcap_{j\in I} P_k(L_j).$$

A restriction on the projection operation that will be needed later is that the projection also be an L_m-observer. From Wong et al. (1995), a characterisation of the *L_m-observer property* is given by the following definition:

Definition 2: Let there be a natural projection $P_i : \Sigma^* \to \Sigma_i^*$ with languages $L_m \subseteq L \subseteq \Sigma^*$. Then P_i is an L_m-observer for L if:

$$(\forall s \in L)(\forall t \in \Sigma_i^*) P_i(s)t \in P_i(L_m) \Longrightarrow (\exists u \in \Sigma^*)$$
$$\text{such that } su \in L_m, \text{ and } P_i(su) = P_i(s)t.$$

If in the above L_m is replaced by L, then it is said that P_i simply has the *observer property* (Wong and Wonham 1996). Intuitively, to have the observer property means that any branching of the system can be observed in the abstracted version of the system. The idea behind the observer property is that any control based on the abstracted model has the same intended effect on the actual plant. Strictly speaking, a projection P_i could hide a branching and still have the observer property if all paths of the branching had the same observed future. In this manner, even though there was branching, the control action would be the same no matter which branch was taken. The L_m-observer property requires that not only does any branching in the marked language have the same observed future, but also that the futures have the same marking.

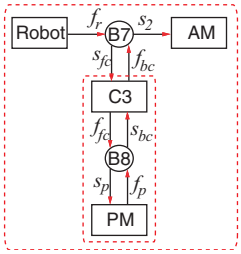

Figure 3. Portion of the flexible manufacturing system example.

The observer property is maintained across synchronous composition in the same manner that the projection operation distributes across synchronous composition. The following result is taken from Pena et al. (2006).

Theorem 1: *Using the definitions of Proposition* 1, *if the natural projection P_k is an $L_{m,j}$-observer for L_j for each $j \in I$ and if $\Sigma_{com} \subseteq \Sigma_k$, then P_k is an L_m-observer for $L = \bigcap_{j\in I} L_j$ where $L_m := \bigcap_{j\in I} L_{m,j}$.*

3. Procedure with example

In this section we will outline the procedure by which the set of modular supervisors is generated through application to a portion of the FMS example introduced in §1 and shown here in Figure 3. The plant consists of a *Robot*, a Conveyor (*C*3), a Painting Machine (*PM*) and an Assembly Machine (*AM*). The two buffers connecting the various machines, B_7 and B_8, serve as the specifications for the system where it is desired that the buffers do not underflow or overflow.

The automata models for the different machines are shown in Figure 4. Note that each of the starting events, s_i, are controllable, and each of the finishing events, f_i, are uncontrollable. Furthermore, recall that all automata have the same alphabet Σ, though irrelevant events are not pictured in the figures. The finite state automata models for the buffers are shown in Figure 5. These buffer models are marked for states for which the buffer is full because our approach requires that the specification languages be prefix-closed, that is, the supervisor cannot change the marking of the uncontrolled plant. If it is necessary to ensure that the system reaches a state where the buffers are empty, another specification could be added.

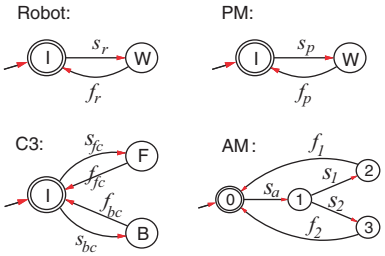

Figure 4. Finite state automaton model of each machine.

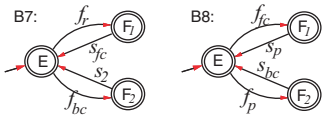

Figure 5. Finite state automaton model of each buffer.

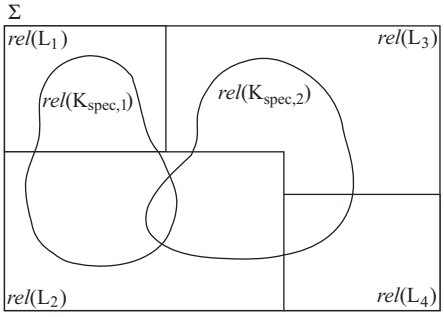

Figure 6. Example relationship between relevant event sets.

We will first present the procedure, and then go through the example. Each series of languages will be indexed sequentially. Furthermore, this procedure and the proofs to follow will assume only one specification is addressed per level of the hierarchy in order to simplify the notation.

3.1 *Supervisor construction procedure*

Input: $L_{m,1}, \ldots, L_{m,n}$, $K_{\text{spec},1}, \ldots, K_{\text{spec},p}$

(1) *Choose an initial specification*
Pick a specification $K_{\text{spec},1}$ and group it with all plant submodules $\{L_1, L_2, \ldots, L_{i_1}\}$ with which it shares a relevant event. We will assume that all plant submodules have already been organised such that they do not share relevant events with each other; all interaction takes place through the specifications. Composing system components so that the plant submodules are disjoint from one another can lead to exponential growth, though in general on a much smaller scale than building the monolithic system. Figure 6 illustrates an example relationship between the relevant event sets of the various plant and specification components in the context of the global event set Σ.

(2) *Perform abstraction*
Perform a projection on the languages generated by the plant submodules from the previous step. The subscript a is used to represent these abstracted languages; specifically, let $L_{i,a} = P_1(L_i)$ and $L_{m,i,a} = P_1(L_{m,i})$ where $i \in \{1,2,\ldots,i_1\}$. Only those events that are not relevant to any specifications on the current or remaining levels of the hierarchy may be considered for erasure. Also, the respective $L_{m,i}$-observer property must be maintained for each of the subplants on the current or remaining levels of hierarchy.

On the first level of the hierarchy, P_1 is therefore defined to maintain the observer property with respect to all $L_{m,i}$. This means that all events that are relevant to only a single component and that do not violate the respective observer properties will be erased by P_1. In practice however, in this first step we will actually only consider those languages with indices in $\{1,2,\ldots,i_1\}$. Those events that do not qualify for erasure make up the set Σ_1 and the associated projection is defined $P_1 : \Sigma^* \to \Sigma_1^*$. Since we have not examined those languages outside the set $\{L_1, L_2, \ldots, L_{i_1}\}$, we do not yet know the exact composition of Σ_1.

(3) *Compose subplant members*
Perform a synchronous composition of all abstracted plant submodules on this level of hierarchy.

$$L'_{1,a} = L_{1,a} \| L_{2,a} \| \cdots \| L_{i_1,a}.$$

Since no relevant events are shared amongst the individual L_i, the projection P_1 distributes across the synchronous composition by Proposition 1. Therefore, $L'_{1,a} = \|_{i \in I_1} P_1(L_i) = P_1(\|_{i \in I_1} L_i)$, where $I_1 = \{1,2,\ldots,i_1\}$. Furthermore, $L'_{1,a} = P_1(L'_1)$ where $L'_1 = \|_{i \in I_1} L_i$. The projection P_1 also possesses the $L'_{m,1}$-observer property by Theorem 1, where $L'_{m,1} = \|_{i \in I_1} L_{m,i}$. Both L'_1 and $L'_{m,1}$ have alphabets equal to the global event set Σ and all abstracted languages on this level of the hierarchy have alphabets equal to Σ_1.

Recall, there are some events in Σ_1 which are as of yet unknown. These events however are not relevant to L'_1, and as such do not cause a

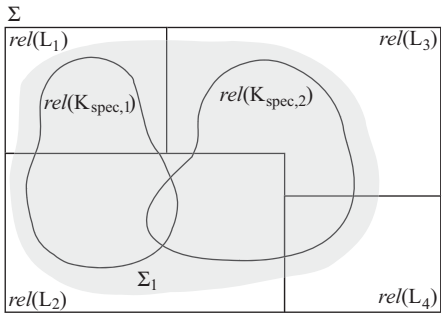

Figure 7. Example relationship between relevant event sets; Σ_1 is the shaded region.

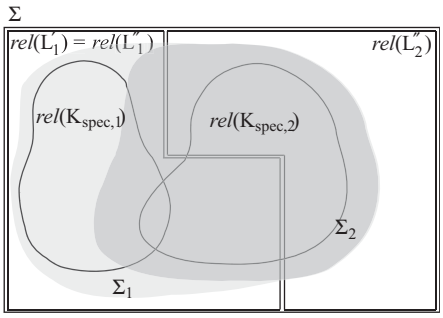

Figure 8. Example relationship between relevant event sets; Σ_2 is the darkly shaded region.

violation of the $L'_{m,1}$-observer property if they are projected away.

The same projection P_1 used in generating the abstracted modular plant $L'_{1,a}$, is also used to generate the corresponding abstracted specification, $K_{\text{spec},1,a} = P_1(K_{\text{spec},1})$. In our approach, events are only considered for abstraction if they are not relevant to the current or remaining specifications, therefore P_1 erases only irrelevant events from $K_{\text{spec},1}$. A representation of the event set Σ_1 can be seen in Figure 7.

(4) *Build a supervisor for each abstracted module*
Following the traditional techniques for supervisor construction, build a nonblocking supervisor for the abstracted plant/specification pair. The component allowable language is defined as $K'_1 = \overline{K_{\text{spec},1}} \cap L'_{m,1}$ and its corresponding abstraction is denoted $K'_{1,a}$.

$$\hat{K}'_{1,a} = \sup\mathcal{C}(K'_{1,a}, L'_{1,a}) \quad \text{where } K'_{1,a} = \overline{K_{\text{spec},1,a}} \cap L'_{m,1,a}. \tag{6}$$

In the expression for $K'_{1,a}$, the projection distributes across the intersection because P_1 does not erase any relevant events from $\overline{K_{\text{spec},1}}$.

In the worst case, the resulting supervisor can be the empty set. However, this supervisor is not guaranteed to be optimal because of the abstraction. Therefore, it is possible that a less restrictive supervisor can be found by erasing fewer events (by making Σ_1 larger). A discussion of how to choose which events to retain can be found at the end of § 4.

(5) *Lift abstracted supervisor to the global level*
In order to apply a supervisor generated in step 4 to the global plant, a default control is implemented that enables all events that had been previously projected away. This is accomplished by the inverse projection operation and is represented by $P_1^{-1}(\hat{K}'_{1,a})$.

(6) *Move up to the next level of the hierarchy*
In order to address those specifications for which a modular supervisor has not yet been built, we now move up to the next level of the hierarchy and repeat steps 1–5. The language representing the closed-loop behaviour of the subsystem from the previous level $\hat{K}'_{1,a}$ becomes a subplant on this level. For example, the allowable language K'_2 for specification $K_{\text{spec},2}$ will be designed with respect to a 'plant' L'_2, that is composed of $\hat{K}'_{1,a}$ from the first level as well as the subplants L_i that were not employed in the first level and that share relevant events with $K_{\text{spec},2}$. Even though on this level $\hat{K}'_{1,a}$ is treated as a subplant, we will still refer to it in terms of its original name which is more consistent with its role on the first level as an allowable language.

Before the synchronous composition is performed, it is again necessary to perform an abstraction $P_2 : \Sigma^* \to \Sigma_2^*$ on each of the submodules such that the observer properties are maintained with respect to any of the plant components on the current or remaining levels of hierarchy. In practice however, we will again only consider those languages $\{L_{i_1+1}, \ldots, L_{i_2}\}$ on this level of the hierarchy. Therefore, we end up with a set consisting of abstracted languages $L_{i,a} = P_2(L_i)$ where $i \in \{i_1 + 1, \ldots, i_2\}$.

$$L'_{2,a} = P_2(\overline{\hat{K}'_{1,a}}) \| L''_{2,a} \quad \text{where } L''_{2,a} = L_{i_1+1,a} \| \cdots \| L_{i_2,a}. \tag{7}$$

Building our 'plant' in this manner is useful in that it causes the new supervisor to be nonconflicting with the supervisor from the previous level. Figure 8 redraws Figure 7 in terms of the newly defined L''_j languages.

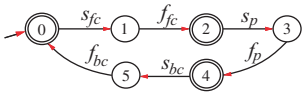

Figure 9. Automaton representing the first modular supervisor which marks the language $\hat{K}'_{1,a}$.

The subplants $\{L_{i_1+1}, \ldots, L_{i_2}\}$ that go into generating $L''_{2,a}$ do not share any relevant events with the specification from the first level. Also, by definition the projection P_2 does not erase events relevant to specifications on the current or higher levels of the hierarchy. Therefore, the only relevant events P_2 erases from $\{L_{i_1+1}, \ldots, L_{i_2}\}$ are not relevant to any component specifications. More specifically, the relevant events P_2 erases from $\{L_{i_1+1}, \ldots, L_{i_2}\}$ are the same events erased by P_1, we just did not know it at the time.

The relevant events erased from the component $\hat{K}'_{1,a}$, however, can be relevant to the specification from the first level if they are not relevant to any of the remaining specifications. Furthermore, events that are relevant only to $\{L_1, L_2, \ldots, L_{i_1}\}$ that violated the respective observer property on the first level can also be reconsidered for erasure. As such, each time we move up a level of the hierarchy, more abstraction is possible, that is $\Sigma_2 \subseteq \Sigma_1$. Figure 8 shows how the set Σ_2 fits into the larger picture.

The behaviour allowed by the supervisor on this level of the hierarchy is determined in the same manner as (6), $\hat{K}'_{2,a} = \sup\mathcal{C}(K'_{2,a}, L'_{2,a})$.

The logic outlined above tells us that the set of relevant events erased from $\{L_{i_1+1}, \ldots, L_{i_2}\}$ by P_2 is disjoint from the set of relevant events erased from $P_1^{-1}(\hat{K}'_{1,a})$. This disjointness will be used in the proofs of §4 to define the projection P_2 as occurring in two stages. Furthermore, since all the components making up L'_2 have disjoint sets of relevant events, the projection still distributes by Proposition 1.

Therefore, $L'_{2,a} = P_2(L'_2)$ where

$$L'_2 = \overline{\hat{K}'_{1,a}} \| L''_2 = P_1^{-1}(\overline{\hat{K}'_{1,a}}) \cap L''_2$$
$$\text{where } L''_2 = L_{i_1+1} \| \cdots \| L_{i_2}. \tag{8}$$

Furthermore, P_2 possesses the $L'_{m,2}$-observer property by Theorem 1, where $L'_{m,2} = P_1^{-1}(\hat{K}'_{1,a}) \cap L''_{m,2}$ and $L''_{m,2} = L_{m,i_1+1} \| \cdots \| L_{m,i_2}$.

(7) *Repeat until finished*

Continue to repeat this process until there are no more specifications left.

Output: $\hat{K}'_{1,a}, \ldots, \hat{K}'_{1,p}$.

Now we present the procedure through a brief FMS example.

3.2 *Supervisor construction procedure with example*

(1) *Choose an initial specification*
We will choose $L_{B8} = \mathcal{L}(B_8)$ to be our first specification where the subplants that it shares relevant events with are $L_{C3} = \mathcal{L}(C3)$ and $L_{PM} = \mathcal{L}(PM)$. Note, the order in which we address specifications and the controlled behaviour that results is not unique.

(2) *Perform abstraction*
The plant submodules comprising the first level of our hierarchy, $C3$ and PM, do not have any events that are not relevant to the current or remaining specifications, thus $L_{C3,a} = P_1(L_{C3})$ and $L_{PM,a} = P_1(L_{PM})$ where P_1 erases only irrelevant events from the languages. In practice, irrelevant events do not need to be projected or added, for the purposes of the proofs to follow however, it is useful to have sets of languages over the same alphabet.

(3) *Compose subplant members*
Composing the individual subplants gives us the plant and specification for our first module, $L'_{1,a} = L_{C3,a} \| L_{PM,a}$ and $\overline{K_{\text{spec},1,a}} = P_1(L_{B8})$ respectively.

(4) *Build a supervisor for each abstracted module*
The abstracted allowable language for the first specification is $K'_{1,a} = L_{B8,a} \cap L_{m,C3,a} \cap L_{m,PM,a}$. It turns out $K'_{1,a}$ is not Σ_u-controllable and hence the supremal controllable sublanguage must be taken. The resulting language is $\hat{K}'_{1,a} = \sup\mathcal{C}(K'_{1,a}, L'_{1,a})$.

(5) *Lift abstracted supervisor to the global level*
This is accomplished by the inverse projection operation. An automaton which generates $\hat{K}'_{1,a}$ and represents the first modular supervisor for our example is shown in Figure 9. Since we do not picture irrelevant events, the events added by the inverse projection are not shown in Figure 9.

(6) *Move up to the next level of the hierarchy*
For our example, we move up a level of hierarchy to build a supervisor for the second specification $L_{B7} = \mathcal{L}(B_7)$. Our new grouping consists of the remaining machines, $L_R = \mathcal{L}(Robot)$ and $L_{AM} = \mathcal{L}(AM)$, as well as the supervised language from the first level $\hat{K}'_{1,a}$. At this point, those events not relevant to the remaining specification are s_r, s_a, s_1, f_1, f_2, s_p, f_p, f_{fc} and s_{bc}. It is interesting to note that the last four events in the above list were relevant

to the first specification and as such could not be considered for erasure on the first level of the hierarchy. We must first however check to see if abstraction of any of these events will cause a violation of the individual $L_{m,i}$-observer properties that we need to guarantee nonblocking. Examination of Figure 4 and Figure 9 will help us to do this.

Specifically, abstraction of the event s_r causes a problem because it represents a transition from a marked state to an unmarked state without an unobservable string of transitions back to a marked state. In other words, if s_r is unobservable then $P_2(s_r) = \varepsilon$ is an element of $P_2(L_{m,R})$, but there is no string u such that $s_r u \in L_{m,R}$ and $P_2(s_r u) = \varepsilon$. Applying similar logic, event s_{bc} cannot be abstracted either. Therefore, $\Sigma_2 = \{s_{fc}, f_{bc}, s_{bc}, s_r, f_r, s_2\}$. We also now know, $\Sigma_1 = \Sigma_2 \cup \{s_p, f_p, f_{fc}\}$. The resulting reduction of machine AM has a single state and generates the language $L_{AM,a} = P_2(L_{AM})$. An automaton which generates the reduction $P_2(\hat{K}'_{1,a})$ has 3 states and 3 transitions. The *Robot* subplant and specification B_7 are not reduced by the projection P_2. That is, $L_{R,a} = P_2(L_R)$ and $L_{B7,a} = P_2(L_{B7})$ where P_2 erases only irrelevant events.

Therefore, the plant for the second specification is $L'_{2,a} = P_2(\hat{K}'_{1,a}) \cap L_{R,a} \cap L_{AM,a}$. Following step 4, the abstracted allowable language is then $K'_{2,a} = L_{B7,a} \cap L'_{m,2,a}$. This language again fails to be Σ_u-controllable. Therefore, taking the supremal controllable sublanguage, the new allowable language for the second specification is $\hat{K}'_{2,a} = \sup\mathcal{C}(K'_{2,a}, L'_{2,a})$. A minimal automaton generator for this language has 8 states and 8 transitions. The final step again is to lift this abstraction up to the global level. Specifically, the events s_a, s_1, f_1, f_2, s_p, f_p, and f_{fc} must be added.

(7) *Repeat until finished*
Since there are no specifications left, we are done.

For the purposes of comparison, the monolithic solution of our example generates a supervisor whose automaton representation consists of 68 states and 157 transitions, while the two modular supervisors are significantly smaller. Specifically, a minimal automaton which generates $\hat{K}'_{1,a}$ has 6 states and 6 transitions and a minimal automaton which generates $\hat{K}'_{2,a}$ has 8 states and 8 transitions (irrelevant events are not considered when counting transitions). Even though the highest level supervisor is built in essence with respect to the full plant, it will almost always be significantly smaller than the monolithic supervisor since it is built to address only those strings relevant to the last specification, rather than the conjunction of all the specifications. It will be shown in § 5 the improvement in complexity can be more dramatic in systems larger than the simple one used here for illustrative purposes.

It is interesting to note that if the modular supervisor for B_7 had been built first and the supervisor for B_8 been built second, the modular supervisors would have had 8 and 10 states and 11 and 13 transitions respectively. Some guidelines for how to best choose the ordering in which the specifications are addressed will be discussed in § 5.

In this case, it turns out that the behaviour allowed by the two modular supervisors is identical to the behaviour allowed by the single monolithic supervisor and hence is optimal. This will not be the case in general since each of the modular supervisors were designed with respect to an abstracted plant. Some discussion of optimality can be found at the end of § 4.

The procedure presented above assumed one specification is addressed per level of hierarchy. This approach however is still valid if multiple specifications are addressed per level, though it will be demonstrated later that it is generally computationally advantageous to address only a single specification. If multiple specifications are addressed within a level, then they must be chosen to address a disjoint set of subplants. The disjointness is needed to provide non-conflict.

4. Controllability and nonblocking

In this section we will demonstrate that the modular supervisors constructed by the procedure of the previous section meet the given specifications without blocking. The following theorem will be stated without proof. The language $\hat{K}'_j$ is constructed in the same manner as $\hat{K}'_{j,a}$ except no abstraction is employed. Additionally, $K = \overline{K_{\text{spec}}} \cap L_m$ where K_{spec} and L_m are defined as in (5) and $L = L_1 \| \ldots \| L_n$.

Theorem 2: *The set of modular supervisors constructed by the procedure of Section* 3, *without any abstraction*, *will provide the same behaviour as the monolithic supervisor*

$$\bigcap_{j=1}^{p} \hat{K}'_j = \sup\mathcal{C}(K, L).$$

The above result states that the conjunction of the modular supervisors built without abstraction will provide the same behaviour as the monolithic supervisor. Equivalence to the monolithic solution implies the conjunction of these modular supervisors provides the optimal solution in the sense of being least restrictive. It further implies that the conjunction of modular supervisors meets the given specifications in a nonblocking manner since the monolithic supervisor does. The problem with this result is that if no abstraction is employed, the procedure essentially requires that the full monolithic system be built. In the following we generate results with abstraction included.

In the context of our approach, we would like to abstract away those elements of our system that are not relevant to those specifications being addressed at the current or higher levels of our hierarchy. For our abstraction, we will simply apply the natural projection operation defined earlier by (2), where it is further required that each projection P_j possesses the $L'_{m,j}$-observer property. The $L'_{m,j}$-observer property is needed for maintenance of nonblocking, but does not provide for optimality.

In order for the supervisor corresponding to the abstracted language to be applied to the global plant, it is necessary to incorporate a default control that permanently enables all events that had been previously hidden. From an implementation standpoint, this can be achieved by the inverse projection, $\tilde{K}_j = P_j^{-1}(\hat{K}_{j,a})$. In the definitions given below, the restriction to $\tilde{K}_{j-1} \cap L''_j$ is added to reflect the actual behaviours of the system that can occur and to assist with the proofs that are to follow. In practice however, these restrictions do not have to be implemented because strings outside the uncontrolled behaviour of the plant and outside the behaviour allowed by other modular supervisors will not occur anyway.

$$\begin{aligned} \tilde{K}_j &= P_j^{-1}(\overline{\hat{K}'_{j,a}}) \cap \tilde{K}_{j-1} \cap L''_j \\ \tilde{K}_{m,j} &= P_j^{-1}(\overline{\hat{K}'_{j,a}}) \cap \tilde{K}_{m,j-1} \cap L''_{m,j}. \end{aligned} \tag{9}$$

In the proofs that follow, it is necessary to show that if an abstracted admissible language $\hat{K}_{j,a}$ is $\Sigma_{u,j}$-controllable with respect to $P_j(L'_j)$, then the lifted version $\tilde{K}_j$ is Σ_u-controllable with respect to the unabstracted language L'_j. Note, $\Sigma_{u,j} = \Sigma_u \cap \Sigma_j$. Furthermore, we need to show that if the supervisor for the abstracted system provides nonblocking control, then the lifted supervisor provides nonblocking control of the actual system.

From examination of the definition of the lifted languages in (9), one can see that they are contained in one another, thereby maintaining nonconflict by construction. Furthermore, it can be shown $\tilde{K}_{m,j} = \tilde{K}_j \cap L_m$.

The following propositions will be needed to prove our desired result. The first proposition is a result from Brandin et al. (2004) which is useful in showing that if an allowable language is Σ_u-controllable with respect to a subset of plant subsystems, it is Σ_u-controllable with respect to the full plant.

Proposition 2: *Let K, $L \subseteq L' \subseteq \Sigma^*$ be languages. Also let $\Sigma_u \subseteq \Sigma$. If K is Σ_u-controllable with respect to L', then K is Σ_u-controllable with respect to L.*

This next proposition shows that the intersection of two nonconflicting Σ_u-controllable languages is itself Σ_u-controllable. The result is in general well-known and can be found in Cassandras and Lafortune (1999).

Proposition 3: *Let K_1, K_2, $L \subseteq \Sigma^*$ be languages and let $K = K_1 \cap K_2$. Also let $\Sigma_u \subseteq \Sigma$. If K_1 and K_2 are nonconflicting and Σ_u-controllable with respect to L, then K is Σ_u-controllable with respect to L.*

Since we are building each successive level of allowable languages with respect to an uncontrolled plant intersected with a controlled language from the previous level, this next proposition is used to show controllability of the intersection of languages from successive levels of our hierarchy.

Proposition 4: *Let K_1, K_2, $L \subseteq \Sigma^*$ be languages and let $K = K_1 \cap K_2$. Also let $\Sigma_u \subseteq \Sigma$ and K_1 and K_2 be nonconflicting. If K_2 is Σ_u-controllable with respect to $\overline{K_1} \cap L$, and K_1 is Σ_u-controllable with respect to L, then K is Σ_u-controllable with respect to L.*

Proof:

$$\begin{aligned} &\overline{K_2}\Sigma_u \cap (\overline{K_1} \cap L) \subseteq \overline{K_2} \\ &\Rightarrow \overline{K_2}\Sigma_u \cap (\overline{K_1} \cap L) \subseteq \overline{K_1} \cap \overline{K_2} \\ &\Rightarrow \overline{K_2}\Sigma_u \cap ((\overline{K_1}\Sigma_u \cap L) \cap L) \subseteq \overline{K_2}\Sigma_u \cap (\overline{K_1} \cap L) \subseteq \overline{K_1} \cap \overline{K_2} \\ &\Rightarrow (\overline{K_2} \cap \overline{K_1})\Sigma_u \cap L \subseteq \overline{K_1} \cap \overline{K_2} \\ &\Rightarrow (\overline{K_1 \cap K_2})\Sigma_u \cap L \subseteq \overline{K_1 \cap K_2}. \end{aligned}$$

□

The following proposition will be used in showing the maintenance of controllability properties between abstracted and lifted languages. It is proven using logic taken from Lin and Wonham (1988).

Proposition 5: *Let $P : \Sigma^* \to \Sigma_a^*$ be a natural projection and let $L \subseteq \Sigma^*$ and $K_a \subseteq \Sigma_a^*$ be languages. Also let $\tilde{K}_m = P^{-1}(\overline{K_a}) \cap L$, $\Sigma_u \subseteq \Sigma$ and $\Sigma_{u,a} = \Sigma_u \cap \Sigma_a$. If K_a is $\Sigma_{u,a}$-controllable with respect to $P(L)$, then $\tilde{K}_m$ is Σ_u-controllable with respect to L.*

The proposition given below is employed to show nonblocking of the abstracted system implies nonblocking of the lifted system. The result follows closely from logic in Wong and Wonham (1996).

Proposition 6: *Let $P : \Sigma^* \to \Sigma_a^*$ be a natural projection and let $\overline{L_m} = L \subseteq \Sigma^*$ and $K_a \subseteq \Sigma_a^*$ be languages. Also let $\tilde{K} = P^{-1}(\overline{K_a}) \cap L$ and $\tilde{K}_m = P^{-1}(\overline{K_a}) \cap L_m$. If the projection P possesses the L_m-observer property and $K_a \subseteq P(L_m)$, then $\overline{\tilde{K}_m} = \tilde{K}$.*

This next proposition demonstrates that if the marking of an abstracted language is consistent with the marking of $P(L_m)$, then the marking of the lifted language will be consistent with the marking of L_m.

Proposition 7: *Let $P : \Sigma^* \to \Sigma_a^*$ be a natural projection and let $\overline{L_m} = L \subseteq \Sigma^*$ and $K_a \subseteq \Sigma_a^*$ be languages. If K_a is $P(L_m)$-closed, then $P^{-1}(\overline{K_a}) \cap L_m = P^{-1}(K_a) \cap L_m$.*

Proof:

$$\begin{aligned}
\overline{K_a} \cap P(L_m) &= K_a \\
P^{-1}(\overline{K_a} \cap P(L_m)) &= P^{-1}(K_a) \\
P^{-1}(\overline{K_a}) \cap P^{-1}(P(L_m)) &= P^{-1}(K_a) \\
P^{-1}(\overline{K_a}) \cap P^{-1}(P(L_m)) \cap L_m &= P^{-1}(K_a) \cap L_m \\
P^{-1}(\overline{K_a}) \cap L_m &= P^{-1}(K_a) \cap L_m.
\end{aligned}$$

□

This final proposition is quite important and will be used to show that languages from successive levels of our hierarchy are nonconflicting.

Proposition 8: *Let $P : \Sigma^* \to \Sigma_a^*$ be a natural projection and let K_1, $K_2 \subseteq \Sigma^*$ be languages. Also let $\mathrm{rel}(K_2) \subseteq \Sigma_a$. If $P(K_2) \subseteq P(K_1)$ and P has the K_1-observer property, then K_1 and K_2 are nonconflicting.*

Proof: Let there be a string $s \in \Sigma^*$ such that $s \in \overline{K_1} \cap \overline{K_2}$. We must then show that K_1 and K_2 share a completion of this string. Since $rel(K_2) \subseteq \Sigma_a$, $\exists t \in \Sigma_a^*$ such that $st \in K_2$. Also since $rel(K_2) \subseteq \Sigma_a$, $P(s)t \in P(K_2) \subseteq P(K_1)$. By the K_1-observer property, $\exists u$ such that $su \in K_1$ and $P(su) = P(s)t$. It then follows that $su \in P^{-1}(P(su)) \subseteq P^{-1}(P(K_2)) = K_2$ since $rel(K_2) \subseteq \Sigma_a$. Therefore, K_1 and K_2 are nonconflicting. □

The propositions provided above will be used to prove the main results of this paper. Before we demonstrate these results, however, we need to examine the exact manner in which our languages are projected and then lifted back to the global alphabet.

In the procedure of § 3, the projection operation $P_2 : \Sigma^* \to \Sigma_2^*$ was defined. This projection was constructed so that it maintained the observer property with respect to each of the elements that went into constructing L_2', specifically, with respect to $\hat{K}_{1,a}'$ as well as each of the elements of $\{L_{i_1+1}, \ldots, L_{i_2}\}$. Since each of these component languages have disjoint sets of relevant events, P_2 has the $L_{m,2}'$-observer property by Theorem 1. It was also stated that P_2 does not erase any relevant events from the language $L_2'' = L_{i_1+1} \| \ldots \| L_{i_2}$ that would not also be erased by the projection $P_1 : \Sigma^* \to \Sigma_1^*$. Furthermore, the events erased by P_2 which are relevant to $\hat{K}_{1,a}'$ are disjoint from those which are relevant to L_2''.

These facts are true for the projection $P_j : \Sigma^* \to \Sigma_j^*$ in general. As such, in the proofs of the following lemmas, it will be assumed that each projection P_j will be performed in two stages. That is, the projection P_1 will be performed first erasing the relevant events of L_j'' followed by a projection $P_j' = \Sigma_1^* \to \Sigma_j^*$ which erases the events relevant to $\hat{K}_{j,a}'$. It is well known that the natural projection satisfies a chain rule property such that $P_j = P_j' \circ P_1$. The proofs to follow will also implement the inverse projection P_j^{-1} using the same two stages. Specifically, $P_j'^{-1}$ will be applied throughout the proofs to generate an intermediate lifted language, $\tilde{K}_{j,a}$. The remaining events will be added separately via the P_1^{-1} operation at the end leading to $\tilde{K}_j$.

By Proposition 1, projection distributes across synchronous composition if no shared relevant events are abstracted away. Since each $L_{m,j}''$ do not share any relevant events, we can therefore define $\tilde{L}_a = P_1(L)$ and $\tilde{L}_{m,a} = P_1(L_m)$. Two other languages projected to the alphabet Σ_1 are defined $\tilde{L}_{m,j,a}' = P_1(L_{m,j}')$ and $\tilde{L}_{m,j,a}'' = P_1(L_{m,j}'')$. These newly defined languages can also be interpreted as versions of the abstracted languages with alphabets equal to Σ_j lifted to the event set Σ_1, rather than all the way to the global alphabet Σ. Figure 10 helps to illustrate the relationship between these newly defined languages.

In a similar manner to the expressions of (9), restrictions have been added to the following definitions of $\tilde{K}_{j,a}$ in order to assist with the proofs that are to follow.

$$\begin{aligned}
\tilde{K}_{j,a} &= P_j'^{-1}(\overline{\hat{K}_{j,a}'}) \cap \tilde{K}_{j-1,a} \cap \tilde{L}_{j,a}'' \\
\tilde{K}_{m,j,a} &= P_j'^{-1}(\overline{\hat{K}_{j,a}'}) \cap \tilde{K}_{m,j-1,a} \cap \tilde{L}_{m,j,a}''.
\end{aligned} \tag{10}$$

Iteratively expanding the terms for the lower level supervisors, $\tilde{K}_{j-1,a}$, leads to the following:

$$\begin{aligned}
\tilde{K}_{j,a} &= \bigcap_{i=1}^{j} P_i'^{-1}(\overline{\hat{K}_{i,a}'}) \cap \bigcap_{i=1}^{j} \tilde{L}_{i,a}'' \\
\tilde{K}_{m,j,a} &= \bigcap_{i=1}^{j} P_i'^{-1}(\overline{\hat{K}_{i,a}'}) \cap \bigcap_{i=1}^{j} \tilde{L}_{m,i,a}''.
\end{aligned} \tag{11}$$

Also note the following definitions of $\tilde{L}_{j,a}'$ and $\tilde{L}_{m,j,a}'$ that also correspond to their unabstracted counterparts.

$$\tilde{L}_{j,a}' = P_{j-1}'^{-1}(\overline{\hat{K}_{j-1,a}'}) \cap \tilde{L}_{j,a}''$$

$$\tilde{L}_{m,j,a}' = P_{j-1}'^{-1}(\hat{K}_{j-1,a}') \cap \tilde{L}_{m,j,a}''.$$

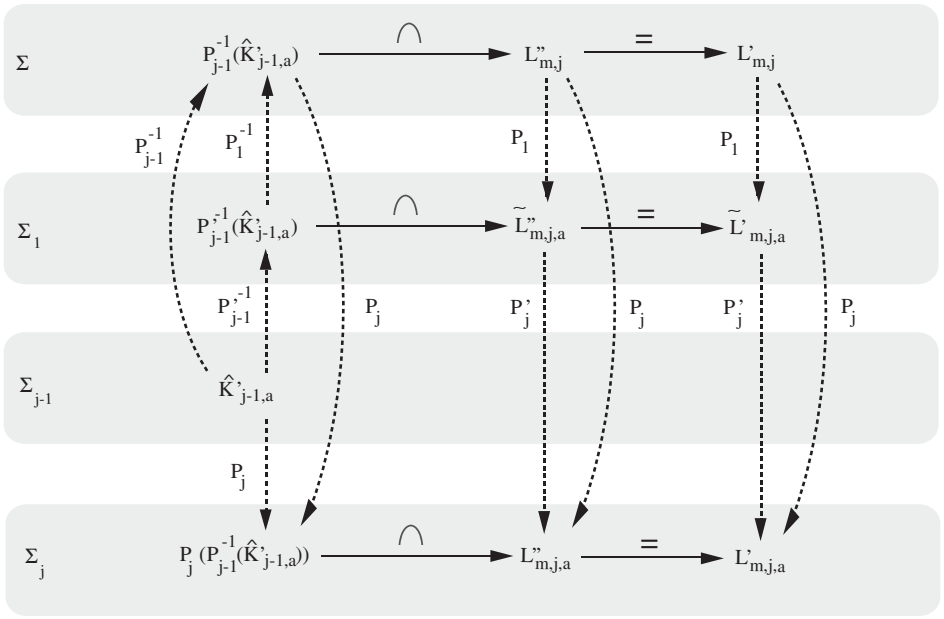

Figure 10. Diagram representing the relationship between various languages and alphabets.

It then follows that $\overline{\tilde{L}'_{m,j,a}} = \tilde{L}'_{j,a}$ since $P'^{-1}_1(\hat{K}'_{j-1,a})$ and $\tilde{L}''_{m,j,a}$ are nonconflicting since they do not share any relevant events. Furthermore, since each $K'_{j,a}$ is by construction $L'_{m,j,a}$-closed, where $L'_{m,j,a} = P'_j(\tilde{L}'_{m,j,a})$, each $\hat{K}'_{j,a}$ is also $P'_j(\tilde{L}'_{m,j,a})$-closed since the sup$\mathcal{C}$ operation will maintain the closure. Therefore, Proposition 7 can be applied iteratively to the above expression for $\tilde{K}_{m,j,a}$ to generate the following:

$$\tilde{K}_{m,j,a} = P'^{-1}_j(\hat{K}'_{j,a}) \cap \tilde{K}_{m,j-1,a} \cap \tilde{L}''_{m,j,a}. \tag{12}$$

Now noting that $\tilde{K}_{j-1,a} \subseteq P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}})$ and $\tilde{K}_{m,j-1,a} \subseteq P'^{-1}_{j-1}(\hat{K}'_{j-1,a})$, (10) can be expressed:

$$\begin{aligned}
\tilde{K}_{j,a} &= P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{K}_{j-1,a} \cap P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}}) \cap \tilde{L}''_{j,a} \\
&= P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{K}_{j-1,a} \cap \tilde{L}'_{j,a} \\
\tilde{K}_{m,j,a} &= P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{K}_{m,j-1,a} \cap P'^{-1}_{j-1}(\hat{K}'_{j-1,a}) \cap \tilde{L}''_{m,j,a} \\
&= P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{K}_{m,j-1,a} \cap \tilde{L}'_{m,j,a}.
\end{aligned} \tag{13}$$

The expressions of (13) will prove useful in the following lemmas.

By definition, P_1 has the $L''_{m,j}$-observer property for all $j \in \{1, \ldots, m\}$. It then follows that since P_1 does not erase any shared relevant events, it will also possess the L_m-observer property by Theorem 2.4. Furthermore, since P_1 erases only irrelevant events from $P^{-1}_{j-1}(\hat{K}'_{j-1,a})$, P'_j will have the observer property with respect to $P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}})$. Since in addition P'_j erases only irrelevant events from $\tilde{L}''_{m,j,a}$, it also possesses the $\tilde{L}'_{m,j,a}$-observer property again by Theorem 1.

We will now present the first main result of the paper, Lemma 1. This lemma shows that the behaviour allowed by the conjunction of modular supervisors constructed by the procedure of §3 is nonblocking. This will be proven by induction, where the induction step shows that the intersection of a sequential set of languages that represent the supervised behaviour of each module is nonconflicting with the supervised behaviour from the next level of the hierarchy.

Lemma 1: *The conjunction of modular supervisors constructed by the procedure of* §3 *is nonblocking if such a set of nonempty modular supervisors exists, that is*

$$\overline{\tilde{K}_{m,1} \cap \tilde{K}_{m,2} \cap \cdots \cap \tilde{K}_{m,p}} = \tilde{K}_1 \cap \tilde{K}_2 \cap \cdots \cap \tilde{K}_p.$$

Proof:

- The first step of this proof is to show that $\overline{\bigcap \tilde{K}_{m,j,a}} = \bigcap \tilde{K}_{j,a}$.
- On the first level of the hierarchy, $\tilde{L}'_{m,1,a} = \tilde{L}''_{m,1,a}$. Furthermore, since P_1 is the projection employed on the first level, $\hat{K}'_{1,a} \subseteq L''_{m,1,a} = \tilde{L}''_{m,1,a}$. Therefore, $\hat{K}'_{1,a} = P'^{-1}_1(\hat{K}'_{1,a})$ and $\tilde{L}''_{m,1,a}$ are nonconflicting and $\overline{\tilde{K}_{m,1,a}} = \tilde{K}_{1,a}$ as built in (10) and (12):

$$\overline{P'^{-1}_1(\hat{K}'_{1,a}) \cap \tilde{L}''_{m,1,a}} = P'^{-1}_1(\overline{\hat{K}'_{1,a}}) \cap \tilde{L}''_{1,a}.$$

- Moving up to successive levels of the hierarchy, each $\tilde{L}'_{m,j,a}$ now consists of a plant subsystem as well as the controlled subplant from the previous level. Since $\tilde{L}'_{m,j,a} = \tilde{L}'_{j,a}$ and $\hat{K}'_{j,a} \subseteq L'_{m,j,a} = P'_j(\tilde{L}'_{m,j,a})$, we can apply Proposition 6,

$$\overline{P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{L}'_{m,j,a}} = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{L}'_{j,a}.$$

- Intersecting both sides of the above with $\tilde{K}_{m,j-1,a} = \tilde{K}_{j-1,a}$ gives us

$$\overline{P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{L}'_{m,j,a}} \cap \overline{\tilde{K}_{m,j-1,a}} = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{L}'_{j,a} \cap \tilde{K}_{j-1,a}. \quad (14)$$

- It is now necessary to show that $P'^{-1}_j(\hat{K}'_{j,a}) \cap \tilde{L}'_{m,j,a}$ and $\tilde{K}_{m,j-1,a}$ are nonconflicting.
- First note the following relation which comes from examination of expressions that are analogous to (6) and (8).

$$\hat{K}'_{j,a} \subseteq K'_{j,a} \subseteq L'_{m,j,a} = P'_j(\tilde{L}'_{m,j,a}) \subseteq P'_j(\tilde{L}''_{m,j,a}).$$

- Also recalling that each P'_j erases only irrelevant events from $\tilde{L}''_{m,j,a}$, the following result holds:

$$P'^{-1}_j(\hat{K}'_{j,a}) \subseteq P'^{-1}_j(P'_j(\tilde{L}''_{m,j,a})) = \tilde{L}''_{m,j,a}. \quad (15)$$

- Taking the prefix-closure of both sides of the above provides the following additional result,

$$P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \subseteq \overline{\tilde{L}''_{m,j,a}} = \tilde{L}''_{j,a}. \quad (16)$$

- Since $K'_{j,a}$ is by construction $P'_j(\tilde{L}'_{m,j,a})$-closed, $\hat{K}'_{j,a}$ is also $P'_j(\tilde{L}''_{m,j,a})$-closed since the sup$\mathcal{C}$ operation will maintain the closure. Therefore, Proposition 7 and (15) can be employed to transform $P'^{-1}_j(\hat{K}'_{j,a}) \cap \tilde{L}'_{m,j,a}$ into

$$P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{L}'_{m,j,a} = P'^{-1}_j(\hat{K}'_{j,a}) \cap (P'^{-1}_{j-1}(\hat{K}'_{j-1,a}) \cap \tilde{L}''_{m,j,a}) = P'^{-1}_j(\hat{K}'_{j,a}) \cap P'^{-1}_{j-1}(\hat{K}'_{j-1,a}). \quad (17)$$

Likewise employing (16),

$$P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{L}'_{j,a} = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap (P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}}) \cap \tilde{L}''_{j,a}) = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}}). \quad (18)$$

Referencing (11) and repeating the logic of (17) and (18):

$$\tilde{K}_{m,j-1,a} = \bigcap_{i=1}^{j-1} P'^{-1}_i(\overline{\hat{K}'_{i,a}}) \cap \bigcap_{i=1}^{j-1} \tilde{L}''_{m,i,a} = \bigcap_{i=1}^{j-1} P'^{-1}_i(\hat{K}'_{i,a}) \quad (19)$$

$$\tilde{K}_{j-1,a} = \bigcap_{i=1}^{j-1} P'^{-1}_i(\overline{\hat{K}'_{i,a}}) \cap \bigcap_{i=1}^{j-1} \tilde{L}''_{i,a} = \bigcap_{i=1}^{j-1} P'^{-1}_i(\overline{\hat{K}'_{i,a}}). \quad (20)$$

- Employing (17), (18), (19) and (20), (14) becomes:

$$\overline{P'^{-1}_j(\hat{K}'_{j,a}) \cap P'^{-1}_{j-1}(\hat{K}'_{j-1,a})} \cap \overline{\bigcap_{i=1}^{j-1} P'^{-1}_i(\hat{K}'_{i,a})} = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}}) \cap \bigcap_{i=1}^{j-1} P'^{-1}_i(\overline{\hat{K}'_{i,a}}). \quad (21)$$

- Now note that $\hat{K}'_{j,a} \subseteq L'_{m,j,a} \subseteq P_j(\hat{K}'_{j-1,a})$. Therefore, $P'_j(P'^{-1}_j(\hat{K}'_{j,a})) \subseteq P'_j(P'^{-1}_{j-1}(\hat{K}'_{j-1,a}))$. Furthermore, $\mathrm{rel}(P'^{-1}_j(\hat{K}'_{j,a})) \subseteq \Sigma_j$. Also since P'_j has the observer property with respect to $P'^{-1}_{j-1}(\hat{K}'_{j-1,a})$, we can apply Proposition 8 to get the following:

$$P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}}) = \overline{P'^{-1}_j(\hat{K}'_{j,a}) \cap P'^{-1}_{j-1}(\hat{K}'_{j-1,a})}.$$

- Since in general, $P'_{j-k+1}(\cap^j_{i=j-k+1} P'^{-1}_i(\hat{K}'_{i,a})) \subseteq P'_{j-k+1}(P'^{-1}_{j-k}(\hat{K}'_{j-k,a}))$, $\mathrm{rel}(\cap^j_{i=j-k+1} P'^{-1}_i(K'_{i,a})) \subseteq \Sigma_{j-k+1}$ and P'_{j-k+1} has the observer property with respect to $P'^{-1}_{j-k}(\hat{K}'_{j-k,a})$, we can repeatedly apply Proposition 8,

$$\bigcap_{i=1}^{j} P'^{-1}_i(\overline{\hat{K}'_{i,a}}) = \overline{\bigcap_{i=1}^{j} P'^{-1}_i(\hat{K}'_{i,a})}. \quad (22)$$

- Comparing the result of (22) to (21) therefore shows that $P'^{-1}_j(\hat{K}'_{j,a}) \cap \tilde{L}'_{m,j,a}$ and $\tilde{K}_{m,j-1,a}$ are nonconflicting. And hence (14) becomes

$$\overline{P'^{-1}_j(\hat{K}'_{j,a}) \cap \tilde{L}'_{m,j,a} \cap \tilde{K}_{m,j-1,a}} = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{L}'_{j,a} \cap \tilde{K}_{j-1,a}.$$

- Then applying Proposition 6 once more,

$$\overline{P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{L}'_{m,j,a} \cap \tilde{K}_{m,j-1,a}} = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \tilde{L}'_{j,a} \cap \tilde{K}_{j-1,a}. \quad (23)$$

Comparing the above to (13) demonstrates that in general, $\tilde{K}_{m,j,a} = \tilde{K}_{j,a}$.

- Furthermore, $\tilde{K}_{m,j,a}$, $\tilde{K}_{m,j-1,a}, \ldots, \tilde{K}_{m,1,a}$ are nonconflicting since $\tilde{K}_{m,j,a} \subseteq \tilde{K}_{m,j-1,a} \subseteq \cdots \subseteq \tilde{K}_{m,1,a}$ by definition. Therefore,

$$\overline{\tilde{K}_{m,1,a} \cap \cdots \cap \tilde{K}_{m,p,a}} = \tilde{K}_{1,a} \cap \cdots \cap \tilde{K}_{p,a}. \quad (24)$$

- The only step left is to address those events abstracted away by P_1.
- Using the definitions in (11), we get the following:

$$\begin{aligned}\bigcap_{j=1}^{p}\tilde{K}_{j,a} &= \bigcap_{j=1}^{p}P_j'^{-1}(\overline{\hat{K}'_{j,a}})\cap\tilde{L}_a\\ \bigcap_{j=1}^{p}\tilde{K}_{m,j,a} &= \bigcap_{j=1}^{p}P_j'^{-1}(\overline{\hat{K}'_{j,a}})\cap\tilde{L}_{m,a}\end{aligned}\tag{25}$$

  therefore, $\bigcap_{j=1}^{p}\tilde{K}_{m,j,a}\subseteq\tilde{L}_{m,a}=P_1(L_m)$.
- Since it is further known that P_1 possesses the L_m-observer property and $\overline{L_m}=L$, Proposition 6 gives us that:

$$\overline{P_1^{-1}\left(\overline{\bigcap_{j=1}^{p}\tilde{K}_{m,j,a}}\right)\cap L_m}=P_1^{-1}\left(\overline{\bigcap_{j=1}^{p}\tilde{K}_{m,j,a}}\right)\cap L.\tag{26}$$

- Recalling (11), (24), (25) and the fact that $L_m\subseteq L\subseteq P_1^{-1}(P_1(L))=P_1^{-1}(L_a)$, we can show the following:

$$\begin{aligned}&P_a^{-1}\left(\overline{\bigcap_{j=1}^{p}\tilde{K}_{m,j,a}}\right)\cap L_m\\ &\quad=P_1^{-1}\left(\bigcap_{j=1}^{p}\tilde{K}_{j,a}\right)\cap L_m\\ &\quad=P_1^{-1}\left(\bigcap_{j=1}^{p}P_j'^{-1}(\overline{\hat{K}_{j,a}})\cap L_a\right)\cap L_m\\ &\quad=\bigcap_{j=1}^{p}P_1^{-1}(P_j'^{-1}(\overline{\hat{K}_{j,a}}))\cap P_1^{-1}(L_a)\cap L_m\\ &\quad=\bigcap_{j=1}^{p}P_j^{-1}(\overline{\hat{K}_{j,a}})\cap L_m=\bigcap_{j=1}^{p}\tilde{K}_{m,j}.\end{aligned}\tag{27}$$

- Similarly, we can show

$$P_1^{-1}\left(\overline{\bigcap_{j=1}^{p}\tilde{K}_{m,j,a}}\right)\cap L=\bigcap_{j=1}^{p}\tilde{K}_j.\tag{28}$$

- Examining (26), (27) and (28), we have shown our desired result, $\tilde{K}_{m,1}\cap\tilde{K}_{m,2}\cap\cdots\cap\tilde{K}_{m,p}=\tilde{K}_1\cap\tilde{K}_2\cap\cdots\cap\tilde{K}_p$. □

Lemma 2 is the second main result of this paper; it shows that the conjunction of languages representing the supervised behaviour of each of the modules is Σ_u-controllable with respect to L. The proof of this result follows the same type of induction logic used in proving Lemma 1.

Lemma 2: *The conjunction of modular supervisors constructed by the procedure of* §3, $\tilde{K}_{m,1}\cap\tilde{K}_{m,2}\cap\cdots\cap\tilde{K}_{m,p}$, *is* Σ_u*-controllable with respect to* L *if such a set of nonempty modular supervisors exists.*

Proof:

- The first step of this proof is to show that $\bigcap\tilde{K}_{m,j,a}$ is $\Sigma_{u,1}$-controllable with respect $L_a=P_1(L)$.
- Beginning on the first level of hierarchy, $\tilde{L}'_{m,1,a}=\tilde{L}''_{m,1,a}$. Therefore, according to (12), $\tilde{K}_{m,1,a}=P_1'^{-1}(\hat{K}'_{1,a})\cap\tilde{L}''_{m,1,a}$.
- Furthermore, since P_1 is the projection employed on the first level, $P_1'^{-1}(\hat{K}'_{1,a})=\hat{K}'_{1,a}$ and $\tilde{L}''_{m,1,a}=L''_{m,1,a}$. Therefore, $\tilde{K}_{m,1,a}=\hat{K}'_{1,a}\cap\tilde{L}''_{m,1,a}$.
- By construction, $\hat{K}'_{1,a}$ is $\Sigma_{u,1}$-controllable with respect to $L''_{1,a}=\tilde{L}''_{1,a}$. Furthermore, since $\hat{K}'_{1,a}\subseteq L''_{m,1,a}=\tilde{L}''_{m,1,a}$, $\hat{K}'_{1,a}$ and $\tilde{L}''_{m,1,a}$ are nonconflicting. Therefore, Proposition 3 provides that $\tilde{K}_{m,1,a}$ is $\Sigma_{u,1}$-controllable with respect to $\tilde{L}''_{1,a}$.
- Furthermore, since $L_a\subseteq\tilde{L}''_{1,a}$, Proposition 2 provides that $\tilde{K}_{m,1}$ is $\Sigma_{u,1}$-controllable with respect to L_a.
- Moving up to higher levels of the hierarchy, each $\hat{K}'_{j,a}$ is $\Sigma_{u,j}$-controllable with respect to $L'_{j,a}=P'_j(\tilde{L}'_{j,a})$ by construction where it is now the case that $\tilde{L}'_{m,j,a}\neq\tilde{L}''_{m,j,a}$. Each $P_j'^{-1}(\hat{K}'_{j,a})\cap\tilde{L}'_{m,j,a}$ is then $\Sigma_{u,1}$-controllable with respect to $\tilde{L}'_{j,a}$ by Proposition 5. $\tilde{L}'_{j,a}$ is built to include a plant subsystem as a well as the controlled subplant from the previous level, $\tilde{L}'_{j,a}=P_{j-1}'^{-1}(\overline{\hat{K}'_{j-1,a}})\cap\tilde{L}''_{j,a}$.
- Applying Proposition 2, $P_j'^{-1}(\overline{\hat{K}'_{j,a}})\cap\tilde{L}'_{m,j,a}$ is $\Sigma_{u,1}$-controllable with respect to $\tilde{K}_{j-1,a}\cap L_a$. Furthermore, based on the logic of Lemma 1, $P_j'^{-1}(\hat{K}'_{j,a})\cap\tilde{L}'_{m,j,a}$ and $\tilde{K}_{m,j-1,a}$ are nonconflicting. Also from the induction hypothesis, $\tilde{K}_{m,j-1,a}$ is $\Sigma_{u,1}$-controllable with respect to L_a. Proposition 4 therefore provides that $P_j'^{-1}(\hat{K}'_{j,a})\cap\tilde{L}'_{m,j,a}\cap\tilde{K}_{m,j-1,a}$ is $\Sigma_{u,1}$-controllable with respect to L_a.
- Examining (13), it can then be seen that each $\tilde{K}_{m,j,a}$ is $\Sigma_{u,1}$-controllable with respect to L_a. Furthermore, since $\tilde{K}_{m,j,a}\subseteq\tilde{K}_{m,j-1,a}\subseteq\ldots\subseteq\tilde{K}_{m,1,a}$, the set of modular supervisors are nonconflicting. Therefore, Proposition 3 provides that the conjunction $\bigcap\tilde{K}_{m,j,a}$ is $\Sigma_{u,1}$-controllable with respect L_a.
- The only step left then is to lift each $\tilde{K}_{m,j,a}$ to the global alphabet.
- Recall, $L_a=P_1(L)$. Since $\bigcap\tilde{K}_{m,j,a}$ is

$\Sigma_{u,1}$-controllable with respect to $P_1(L)$, Proposition 5 provides that $P_1^{-1}(\bigcap \tilde{K}_{m,j,a}) \cap L_m$ is Σ_u-controllable with respect to L. Examination of (27) therefore demonstrates that the behaviour allowed by the conjunction of modular supervisors $\tilde{K}_{m,1} \cap \tilde{K}_{m,2} \cap \cdots \cap \tilde{K}_{m,p}$, is Σ_u-controllable with respect to L. □

Lemma 1 and Lemma 2 then provide us with the overall desired result, that the behaviour provided by the supervisor languages constructed in § 3 is safe and nonblocking. Specifically, Lemma 1 demonstrates that the behaviour is nonblocking and Lemma 2 demonstrates that the control required by the supervisors is realizable.

Theorem 3: *The set of modular supervisors constructed by the procedure of* § 3 *will meet the given specifications in a nonblocking manner when acting in conjunction if such a set of nonempty nonblocking modular supervisors exists.*

Proof: Follows directly from Lemma 1 and Lemma 2. □

The result of Theorem 3 demonstrates that employing abstractions that possess the respective observer properties provides Σ_u-controllability and nonblocking. However, it does not necessarily lead to an optimal solution. This suboptimality arises at least in part from the fact that if a relevant controllable event is abstracted away, the ability to disable it as a means of preventing an uncontrollable continuation is lost.

In the context of a single system, (Zhong and Wonham 1990) introduced a property called output-control-consistency (OCC) that addresses this situation. The presence of this OCC property along with the observer property can be employed to demonstrate that the optimal control generated for an abstracted system can be applied to the unabstracted system and still achieve optimal supervision. Maintenance of the OCC property in modular approaches however is less well understood. For our approach, we will leave the problem of optimal supervision as an open problem. It is however understood that abstracting away fewer controllable events will make the behaviour allowed by our approach less restrictive.

5. Experimental results

In this section we will implement our procedure on two moderately large examples. The purpose of this exercise is to further demonstrate the strengths of our approach, and to provide some heuristics for its implementation. Specifically, guidelines will be provided for how to choose the order in which the modular specifications are addressed. As was demonstrated in the example of § 3, the order in which the modular supervisors are built is not unique and can have a large effect on the size of the resulting supervisors.

5.1 *AGV example*

The first example we will examine concerns the supervision of a system of automated guided vehicles (AGVs). The details of the component specification and plant modules employed here can be found in Wonham (2006). Specifically, there are five plant modules $\{AGV_1, AGV_2, AGV_3, AGV_4, AGV_5\}$ representing each of five AGVs. Also, there are a total of eight component specifications. Five of these $\{Z_1, Z_2, Z_3, Z_4, IPS\}$ represent areas of the factory that can only be occupied by a single AGV at a time, thereby avoiding the collision of AGVs. The remaining three specifications $\{WS_1, WS_2, WS_3\}$ represent workstations and dictate the order in which parts are unloaded from and loaded to the AGVs. The example presented in Wonham (2006) is modified from its original form in Holloway and Krogh (1990). The monolithic supervisor built for this system is represented by an automaton that has 4406 states and 11 338 transitions. Furthermore, when all component specifications and plant modules are composed into a single automaton, that automaton has 22 784 states and 67 520 transitions. Note that traditionally constructed modular supervisors are pairwise nonconflicting, but block when all are acting in conjunction.

One possible solution employing our approach addresses the specifications in the order $Z_1 \rightarrow IPS \rightarrow Z_2 \rightarrow WS_2 \rightarrow Z_3 \rightarrow WS_3 \rightarrow Z_4 \rightarrow WS_1$. In this process, the largest resulting modular supervisor is represented by an automaton that has 96 states and 250 transitions. The largest automaton built throughout this example has 144 states and 302 transitions.

In the above, we have employed a heuristic algorithm based on results in Su and Wonham (2006) for choosing the order in which specifications are addressed. In general, the goal is to first choose specifications which result in small supervisors and supervisors that have the most potential for reduction by abstraction. Furthermore, only a single specification is addressed per level of the hierarchy. If instead, multiple 'disjoint' supervisors are built per level of hierarchy, then the subsystems must be composed if

Table 1. Summary of example results.

Case	# states (# transitions) in largest supervisor	# states (# transitions) in largest intermediate automaton
AGV Monolithic (opt)	*4406* (*11 338*)	*22 784* (*67 520*)
AGV Modular 1 (subopt)	96 (250)	144 (302)
AGV Modular 2 (opt)	576 (1700)	864 (2100)
FMS Monolithic (opt)	*780* (*2491*)	*4048* (*14 320*)
FMS Modular (opt)	25 (37)	41 (80)

they are both addressed by the next specification. This composition process often leads the next supervisor to have a larger state space than if only a single specification had been addressed.

The modular solution developed here turns out to be suboptimal. Specifically, it is more restrictive in terms of which areas of the factory can by occupied by AGVs at a given time than the monolithic solution which is maximally permissive. However, both the modular and monolithic solutions allow all three workstations to contain workpieces simultaneously. In the summary of numerical results presented in Table 1, this solution is referred to as AGV Modular 1. If the procedure of this paper uses the same ordering of specifications, but chooses to not abstract away some of the controllable events that could otherwise be erased, then it too provides optimal control. The largest modular supervisor in this case is significantly larger in that it is represented by an automaton with 576 states and 1700 transitions. Furthermore, the largest intermediate automaton built in this process has 864 states and 2100 transitions. In Table 1, this solution is referred to as AGV Modular 2.

5.2 *FMS example*

We will now provide results for the FMS example first introduced in §1. This example is modified from its original presentation in de Queiroz et al. (2005). Our version specifically consists of fewer components and as such some of the component models have fewer states and smaller event sets than the corresponding models given in de Queiroz et al. (2005). The monolithic supervisor for this example is represented by an automaton with 780 states and 2491 transitions, while the composition of all the machine and buffer models leads to an automaton with 4048 states and 14 320 transitions.

Applying our approach to this example where the specifications are addressed in the order $B_8 \rightarrow B_7 \rightarrow B_4 \rightarrow B_2$ leads to a set of modular supervisors where the largest corresponding automaton has 25 states and 37 transitions. Additionally, the largest automaton built in this process has 41 states and 80 transitions. It turns out for this example that the modular and monolithic solutions provide the same behaviour. In this respect the modular solution is optimal. The numerical details of this example are also presented in Table 1.

5.3 *Discussion of complexity*

Assessment of the complexity advantage provided by the approach of this paper is difficult because in the worst case no abstraction is possible and the size of the state space grows at the same exponential rate as the monolithic approach. As a result, we cannot show a bound on complexity that is guaranteed to give an improvement as compared to the monolithic solution. However, it is generally the case that significant abstraction is possible through the steps of this procedure resulting in a slowing of the exponential growth of the state space. The examples presented in this section specifically give some indication of the level of reduction achievable in terms of the size of the automata that are built. A table summarizing the results of each of the examples of this section is given in Table 1.

Apart from the size of the resulting automata, it is also necessary to consider the complexity of the additional operations required of the procedure of this paper, including, the process of applying a natural projection and the process of finding a natural projection with the observer property. In general, it is known that in the worst case the projection of a DES can lead to an exponential growth of the state space, thereby indicating the time complexity of the operation is at worst exponential. However, it has been demonstrated in Wong (1998) that a projection with the observer property is guaranteed to result in an abstracted system that is no larger than the original system. Furthermore, Wong (1998) demonstrates that under these conditions the complexity of generating the projected model is at worst polynomial in time. As far as finding a projection that possesses the observer property, (Feng 2006) presents a polynomial time

algorithm for finding an extension of the set of observable events for a projection such that the projection is an observer. Since the process of building a supervisory controller also has polynomial complexity, the complexity of the monolithic and modular approaches will both be dominated by a polynomial complexity operation applied to the largest intermediate automaton that has to be built. The size of the largest intermediate automaton therefore is a reasonable metric for the complexity of generating supervisory control for a given system. Furthermore, the size of the largest resulting supervisor provides a reasonable metric for the complexity of the implementation of the control scheme.

It should be noted that the algorithm of Feng (2006) in general finds a reasonably small extension of the observable event set, though it is not guaranteed to be minimal. This fact, along with the problem of ordering, indicates that it is not possible to find the minimal reduction of the state space. We can however use intuition and experience to find reasonably good reductions.

One way to further improve the complexity of our approach is to combine it with other techniques that exist in the literature. For example, the work of Feng and Wonham (2006b) provides results for certain commonly encountered types of systems. For these classes of systems, it is shown that conditions on certain modules indicate that they will be nonconflicting with the rest of the system. As such, these modules do not have to be considered when trying to guarantee nonblocking of the global system. Reduction in the number of components that must be considered directly results in less complexity since the size of the state space grows exponentially in the number of components. It is also possible that the approach of this paper can be employed with more computationally efficient data structures than automata, like BDDs (Bryant 1986) and state tree structures (Ma 2004). In Ma (2004) and Song and Leduc (2006) different data structures are combined with variations of the supervisory control framework to improve complexity and understandability.

6. Conclusions and future work

This paper puts forth an incremental procedure for generating a set of modular supervisors that are nonconflicting by construction. The conjunction of the modular supervisors was shown to satisfy given specifications without blocking when natural projections with the L_m-observer property are employed. It is also shown that in general the set of modular supervisors provide suboptimal control. It is argued that this potential loss of optimality is worth the reduction in complexity this approach provides.

Examples were presented in § 5, with our modular solution showing significant savings in size as compared to the monolithic solution. The approach to supervisor design presented here is generally useful for systems where the coupling between elements is well-distributed. If every specification of a system addresses every plant module, then this approach will likely provide little improvement.

Some directions for future work include further examination of different types of abstraction, state aggregation approaches like observation equivalent (Milner 1989) and conflict equivalent (Malik et al. 2004) reductions are two possibilities. Another improvement would be to determine conditions under which optimality is achieved. A final direction of work would be to employ the algorithms presented here in combination with more efficient data structures than automata.

Acknowledgements

The authors gratefully acknowledge the contribution of Professor Stéphane Lafortune. His involvement in developing and clarifying some of the proofs was invaluable. The authors would also like to thank Professor W.M. Wonham and Dr. Lei Feng for making available software for testing the observer property. This software enabled the addition of the examples of § 5. Finally, the authors thank the anonymous reviewers for their thorough and helpful comments. This work was supported in part by NSF CMS 05-28287.

References

Brandin, B.A., Malik, R., and Malik, P. (2004), "Incremental Verification and Synthesis of Discrete-Event Systems Guided by Counter Examples," *IEEE Transactions on Control Systems Technology*, 12, 387–401.

Bryant, R.E. (1986), "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, C-35, 677–691.

Cassandras, C., and Lafortune, S. (1999), *Introduction to Discrete Event Systems*, Boston, MA: Kluwer Academic Publishers.

de Queiroz M.H., and Cury J.E.R. (2000), "Modular Supervisory Control of Composed Systems", in *Proc. American Control Conf.*, Chicago, USA, pp. 4051–4055.

de Queiroz, M.H., Cury, J.E.R., and Wonham, W. (2005), "Multitasking Supervisory Control of Discrete-Event Systems," *Discrete Event Dynamic Systems: Theory and Application*, 15, 375–395.

Endsley, E., and Tilbury, D. (2004), "Modular Finite State Machines for Logic Control," *Proc. Int. Workshop on*

Discrete Event Systems (WODES), Reims, France, pp. 403–408.

Feng, L. (2006), On the Computation of Natural Observers in Discrete-Event Systems. Technical report, University of Toronto, Systems and Control Group, Toronto, Canada.

Feng, L., and Wonham, W. (2006a), "Computationally Efficient Supervisor Design: Abstraction and Modularity," *Proc. Int. Workshop on Discrete Event Systems (WODES)*, Ann Arbor, USA, pp. 3–8.

Feng, L., and Wonham, W. (2006b), "Computationally Efficient Supervisor Design: Control Flow Decomposition," *Proc. Int. Workshop on Discrete Event Systems (WODES)*, Ann Arbor, USA, pp. 9–14.

Flordal, H., and Malik, R. (2006a), "Modular Nonblocking Verification Using Conflict Equivalence," *Proc. Int. Workshop on Discrete Event Systems (WODES)*, Ann Arbor, USA, pp. 100–106.

Flordal, H., and Malik, R. (2006b), "Supervision Equivalence," *Proc. Int. Workshop on Discrete Event Systems (WODES)*, Ann Arbor, USA, pp. 155–160.

Gaudin, B., and Marchand, H. (2004), "Modular Supervisory Control of a Class of Concurrent Discrete Event Systems," *Proc. Int. Workshop on Discrete Event Systems (WODES)*, Reims, France, pp. 181–186.

Gaudin, B., and Marchand, H. (2005), "Efficient Computation of Supervisors for Loosely Synchronous Discrete Event Systems: A State-Based Approach," in *6th IFAC World Congress*, Prague, Czech Republic.

Hill, R., and Tilbury, D. (2006), "Modular Supervisory Control of Discrete-Event Systems with Abstraction and Incremental Hierarchical Construction," *Proc. Int. Workshop on Discrete Event Systems (WODES)*, Ann Arbor, USA, pp. 398–406.

Holloway, L., and Krogh, B. (1990), "Synthesis of Feedback Logic Control for a Class of Controlled Petri Nets," *IEEE Transactions on Automatic Control*, 35, 514–523.

Komenda J., van Schuppen J., Gaudin B., and Marchand H. (2005), "Modular Supervisory Control with General Indecomposible Specification Languages," in *Proc. 44th IEEE Conf. Decision & Control and European Control Conf.*, Sevilla, Spain, pp. 3474–3479.

Leduc, R.J., Lawford, M., and Wonham, W. (2005), "Hierarchical Interface-Based Supervisory Control Part II: Parallel Case," *IEEE Transactions on Automatic Control*, 50, 1336–1348.

Lee, S.H., and Wong, K.C. (1997), "Decentralised Control of Concurrent Discrete-Event Systems with Non-Prefix Closed Local Specifications," in *Proc. 36th IEEE Conf. Decision & Control*, San Diego, USA, pp. 2958–2963.

Lin, F., and Wonham, W. (1988), "Decentralized Supervisory Control of Discrete-Event Systems," *Information Sciences*, 44, 199–224.

Ma, C. (2004), "Nonblocking Supervisory Control of State Tree Structures." PhD thesis, University of Toronto, Toronto, Canada.

Malik, R., Streader, D., and Reeves, S. (2004) "Fair Testing Revisited: A Process-Algebraic Characterisation of Conflicts," in *Proc. 2nd Int. Symp. Automated Technology for Verification and Analysis*, *ATVA 2004*, pp. 120–134.

Milner, R. (1989), *Communication and Concurrency*, London: Prentice-Hall, Inc.

Pena, P., Cury, J., and Lafortune, S. (2006), "Testing Modularity of Local Supervisors: An Approach Based on Abstractions," *Proc. Int. Workshop on Discrete Event Systems (WODES)*, Ann Arbor, USA, pp. 107–112.

Ramadge, P., and Wonham, W. (1988), "Modular Supervisory Control of Discrete Event Systems," *Mathematics of Control, Signal and Systems*, 1, 13–30.

Ramadge, P., and Wonham, W. (1989), "The Control of Discrete Event Systems," *Proc. of IEEE, Special Issue on Discrete Event Dynamic Systems*, 77, pp. 81–98.

Schmidt, K., Moor, T., and Perk, S. (2005), "A Hierarchical Architecture for Nonblocking Control of Discrete Event Systems," *Mediterranean Conference on Control and Automation*, Limassol, Cyprus, pp. 902–907.

Song, R., and Leduc, R.J. (2006), "Symbolic Synthesis and Verification of Hierarchical Interface-based Supervisory Control," *Proc. Int. Workshop on Discrete Event Systems (WODES)*, Ann Arbor, USA, pp. 419–426.

Su, R., and Wonham, W. (2006), "Hierarchical Fault Diagnosis for Discrete-Event Systems under Global Consistency," *Discrete Event Dynamic Systems: Theory and Application*, 16, 39–70.

Su, R. (2004), "Distributed Diagnosis for Discrete-Event Systems." PhD thesis, University of Toronto, Toronto, Canada.

Wong, K. (1998), "On the Complexity of Projections of Discrete Event Systems," *in Proc. Int. Workshop on Discrete Event Systems (WODES)*, Cagliari, Italy, pp. 201–206.

Wong, K., Thistle, J., Hoang, H.H. and Malhamé R. (1995), "Conflict Resolution in Modular Control With Applications to Feature Interaction," in *Proc. IEEE Conf. on Decision & Control*, New Orleans, USA, pp. 416–421.

Wong, K., and Wonham, W. (1996), "Hierarchical Control of Discrete-Event Systems," *Discrete Event Dynamic Systems: Theory and Application*, 6, 241–273.

Wong, K., and Wonham, W. (1998), "Modular Control and Coordination of Discrete-Event Systems," *Discrete Event Dynamic Systems: Theory and Application*, 8, 247–297.

Wonham, W. (2006), *Supervisory Control of Discrete-Event Systems*, University of Toronto: ECE Dept., current update 2006.07.01. Available at http://www.control.utoronto.ca/DES.

Zhong, H., and Wonham, W. (1990), "On the Consistency of Hierarchical Supervision in Discrete-Event Systems," *IEEE Transactions on Automatic Control*, 35, 1125–1134.