Discrete Event Dynamic Systems: Theory and Applications, 6, 379–427 (1996)

Centralized and Distributed Algorithms for On-Line Synthesis of Maximal Control Policies under Partial Observation

NEJIB BEN HADJ-ALOUANE
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122

STÉPHANE LAFORTUNE stephane@eecs.umich.edu
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122

FENG LIN flin@ece.eng.wayne.edu
Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202

Received November 21, 1994; Revised September 25, 1995

Abstract. This paper deals with the on-line control of partially observed discrete event systems (DES). The goal is to restrict the behavior of the system within a prefix-closed legal language while accounting for the presence of *uncontrollable* and *unobservable* events. In the spirit of recent work on the on-line control of partially observed DES (Heymann and Lin 1994) and on variable lookahead control of fully observed DES (Ben Hadj-Alouane et al. 1994c), we propose an approach where, following each observable event, a control action is computed on-line using an algorithm of *linear* worst-case complexity. This algorithm, called **VLP-PO**, has the following additional properties: (i) the resulting behavior is guaranteed to be a *maximal* controllable and observable sublanguage of the legal language; (ii) different maximals may be generated by varying the priorities assigned to the controllable events, a parameter of **VLP-PO**; (iii) a maximal containing the supremal controllable and normal sublanguage of the legal language can be generated by a proper selection of controllable event priorities; and (iv) no off-line calculations are necessary. We also present a parallel/distributed version of the **VLP-PO** algorithm called **DI-VLP-PO**. This version uses several communicating agents that simultaneously run (on-line) identical versions of the algorithm but on possibly different parts of the system model and the legal language, according to the structural properties of the system and the specifications. While achieving the same behavior as **VLO-PO, DI-VLP-PO** runs at a total complexity (for computation and communication) that is significantly lower than its sequential counterpart.

Keywords: supervisory control, partial observation, variable lookahead policies, on-line control, parallel/distributed algorithms, modular control

1. Introduction

On-line schemes for the control of discrete event systems (DES) have been considered in the recent literature because of the potential that they offer to mitigate the computational burden associated with off-line controller synthesis techniques for large-scale DES. On-line techniques also allow us to deal with incomplete system models, time-varying systems, and infinite state systems (see (Chung et al. 1992), (Ben Hadj-Alouane et al. 1994c)). An off-line scheme is one where the complete control policy for all possible states of the controlled system is calculated off-line and then stored; at run-time, the desired control action at each step (i.e., after each observable event executed by the system) is then retrieved from a look-

up table and enforced. We call a scheme on-line if at each run-time step the control action to be applied is calculated using part or all of the system model and possibly also using stored results from prior calculations. When the DES to be controlled is only partially observed, the need for on-line techniques is even stronger as off-line controller synthesis techniques generally require building an observer for the system model, a procedure which is, in the worst case, of exponential complexity in the cardinality of the system state space. In on-line control, the objective is to calculate the control action at each step with an algorithm of polynomial (preferably linear) complexity. Thus, the exponential complexity of off-line synthesis of a complete control policy is "broken down" into a set of polynomial complexity calculations along the specific trace of events generated by the system at run-time.

This paper deals with the on-line control of partially observed DES. The goal is to restrict the behavior of the system within a prefix-closed legal language (a sublanguage of the system language) while accounting for the presence of *uncontrollable* and *unobservable* events (cf. (Cieslak et al. 1988), (Lin and Wonham 1988), (Ramadge and Wonham 1987) and the survey paper (Ramadge and Wonham 1989)). Our initial work in (Chung et al. 1992) is for completely observed DES and deals with the so-called *limited lookahead* situation where the controller has only access to the next N possible events that the system can execute, and not to the true system state. An efficient forward search algorithm termed **VLP** (for Variable Lookahead Policy algorithm) is proposed for this situation in (Chung et al. 1994); the forward search feature of this algorithm motivated the terminology "variable lookahead." The situation where the controller does have access to the true system state is considered in (Ben Hadj-Alouane et al. 1994c) where the **VLP-S** algorithm (for Variable Lookahead Policy with State information) is developed. Observe that the **VLP-S** algorithm can be used in a limited lookahead context (i.e., the forward search is limited to N steps into the future) as well as in the more general context where no such N is specified. The latter situation is of course bounded too, but by the total number of states in the system, in the worst-case; and since there is no *a priori* bound N, this situation corresponds to an exact on-line implementation of the standard off-line controller synthesis techniques for DES. In the spirit of this work and that of (Heymann and Lin 1994), we propose a new algorithm for the on-line synthesis of control policies for partially observed DES. This algorithm, acronymed **VLP-PO** (for *variable lookahead policies* under *partial observation*), is based on a forward search over the system model. In the "variable lookahead" version of this algorithm, the depth of the search window is not set *a priori*, and if necessary, the resulting control action is determined based on the entire system and legal language models (though we hope that only part of these models is sufficient). **VLP-PO** is called immediately following the execution of a new observable event; it computes, based on the current observer state, the system and legal language models, and previously stored computation, the next control action and the next observer state. This process is repeated after each observable event along the trace of events (*i.e.*, sample path) executed by the system at run-time.

The **VLP-PO** algorithm has the following properties:

(i) The worst-case computational complexity for calculating the next control action is $O(\max(|X||\Sigma_c|^2, |X||\Sigma|))$, where X denotes the state space of the system, Σ the set of events, and Σ_c the subset of controllable events.

(ii) **VLP-PO** always generates a *maximal* controllable and observable sublanguage of the legal language; this guarantees that if the given partial observation control problem has solutions (i.e., the legal language has a non-empty controllable and observable sublanguage) one is generated.

(iii) Hence, if the supremal controllable and observable sublanguage of the legal language exists, it is automatically generated. (Note that this will not be the case for approaches based on the supremal controllable and normal sublanguage, as this language could be empty even if a solution exists.)

(iv) Moreover, one of the parameters given to **VLP-PO** before each control action computation is a total ordering over the subset of controllable events, Σ_c; by varying this parameter, different maximals may be generated. If the ordering of controllable events is carefully selected to reflect some performance measure (such as event priorities) a desirable maximal may be generated.

(v) In addition, **VLP-PO** is guaranteed to generate a maximal that contains the supremal controllable and normal sublanguage of the legal language, if the ordering of the controllable events that is selected satisfies a certain property (see Section 5.2).

(vi) **VLP-PO** works completely on-line; i.e., no off-line calculations, such as finding the supremal controllable sublanguage of the legal language, are necessary.

VLP-PO is best viewed as an extension of the **VLP-S** algorithm of Ben Hadj-Alouane et al. (1994c) to the case of partially-observed systems, but for the special case of prefix-closed legal languages. Properties (ii)–(vi) of **VLP-PO**, listed above, are in contrast with the properties of the algorithms proposed in (Heymann and Lin 1994). In (Cho and Marcus 1989b) an off-line method for generating a maximal observable and controllable sublanguage of the legal language for prefix-closed languages is presented. However, the convergence of this method is still an open question (obviously, if it converges, it incurs an exponential worst-case complexity). By running **VLP-PO** "off-line" (i.e., run it for each state of the observer), we are guaranteed to produce a control policy that will generate a maximal. Moreover, the method in (Cho and Marcus 1989b) does not possess properties (iv) and (v), which provide some flexibility in the selection of the generated maximal, a useful tool in practice.

As mentioned above, in **VLP-PO** no *a priori* bound is imposed on the depth of the forward search needed for the calculation of a control action. If the system is finite-state, the search is guaranteed to terminate. However, the termination of **VLP-PO** when used with an infinite-state system is contingent upon the finiteness of a *given* bound. In this regard, we present and discuss a so-called "limited lookahead version" of **VLP-PO**, where the depth of the search cannot exceed a preset bound. This bound, N, can be used to deal with infinite state systems or to model situations where only partial knowledge about the system model is available at each run-time step (even though the system may be finite-state). The bound can also be used to model situations where a control action has to be returned by a given deadline during the on-line operation of the system.

Next we present a parallel/distributed version of **VLP-PO**, termed **DI-VLP-PO**. We show that by making mild assumptions on the structure of the DES and of the legal language, we

are able to "distribute" the work performed by the **VLP-PO** algorithm over several communicating agents which run concurrently. The resulting **DI-VLP-PO** algorithm preserves all the properties of **VLP-PO** but has a total execution complexity that is significantly less than the sequential **VLP-PO** algorithm. More specifically, the situation under consideration is that of a DES G whose structure is such that it can be expressed as the shuffle of several state machines (with disjoint event sets). The legal language K is global, but it can be expressed as the parallel composition of several sublanguages. The algorithm uses several communicating agents that simultaneously run (on-line) identical versions of the **DI-VLP-PO** algorithm but on possibly different parts of the system model and of the legal language; in fact, there is exactly one agent for each sublanguage constituting the legal language. We show that this scheme achieves the same behavior as the sequential **VLP-PO** algorithm and thus all the properties of **VLP-PO** are preserved. Significant savings in computation time are achieved by the parallel/distributed implementation: The worst-case execution complexity (at each control action computation) of **DI-VLP-PO** can be several orders of magnitude better than its sequential counter-part **VLP-PO** (see Sections 8 and 10 for exact expressions). The maximum number of messages exchanged between the agents of **DI-VLP-PO** (at each control action computation) is of the same order as $|\Sigma_c|^2$ (see Sections 8 and 10 for exact expressions). In many practical applications, the system G does satisfy the above assumption. Even if this is not the case, significant computational savings can still be achieved by **DI-VLP-PO** (see Section 11, where the only assumption is that K is the intersection of several sublanguages).

In (Garg 1993), parallel implementations of existing sequential supervisory control synthesis algorithms are given. However, this work is different from ours because no particular structure of the system or legal language is assumed and the emphasis is on fine grain parallelization. In (Ramadge 1989), system and legal language structures slightly similar to ours are used to reduce the problem complexity (within the context of Büchi automata). However, only full observation supervision is considered, and the complexity reduction is achieved by adopting a "decentralized-like" supervision scheme (not via an on-line distributed/parallel calculation of the control, as we propose here).

Notice that we use the expression parallel/distributed to emphasize that the **DI-VLP-PO** algorithm could be implemented in a "parallel" (or systolic) manner, i.e., each agent would be assigned to a processor in a multiprocessor machine and communication would occur via shared memory or message passing on a common bus; or it could be implemented in a "distributed" manner, i.e., each agent would be assigned to a different workstation and communication would occur via a (fast) communication network. The latter situation is typical of a factory floor for instance.

This paper is organized as follows. In Section 2, we present necessary background material on the supervisory control theory of DES. In Section 3, we recall results from (Ben Hadj-Alouane et al. 1994c) on the on-line control of fully observed DES which are used in this paper. In Section 4 we present the **VLP-PO** algorithm, prove its correctness (in the sense that no illegal behavior can be generated) and determine its worst-case computational complexity. In Section 5 the properties related to the maximality of **VLP-PO** generated policies are proved. In Section 6 a resource allocation problem is presented to illustrate the application of **VLP-PO**. In Section 7 the limited lookahead version of

VLP-PO is discussed. In Section 8 we present the two-agent version of the **DI-VLP-PO** algorithm and establish its properties in terms of correctness, execution complexity and communication complexity (the part of the execution complexity dealing with the exchange of messages). As it turns out this special case possesses all the key features of the general structure, but it can be easily presented and understood since it involves only two agents. Section 9 illustrates the use of the two-agent **DI-VLP-PO** with the resource allocation problem. Section 10 presents the general multi-agent **DI-VLP-PO** and extends the results of the two-agent version to the general case. Section 11 discusses the two special cases of *modular* control and of control under *complete event observations*. Some concluding remarks are made in Section 12. (We note that preliminary versions of some of the results in this paper have been presented, without proof, in (Ben Hadj-Alouane et al. 1993), (Ben Hadj-Alouane et al. 1994a), (Ben Hadj-Alouane et al. 1994b).)

2. Background on Supervisory Control

First, we overview the standard supervisory control problem—SSCP— described in detail in (Ramadge and Wonham 1987). Then, we outline the corresponding partial observation version introduced in (Lin and Wonham 1988).

The discrete event system is modeled by a generator (or state machine) $G = (\Sigma, X, \delta, x_0, X_m)$, where Σ denotes the events, X the state space, δ: $X \times \Sigma \longrightarrow X$ the transition function, x_0 the initial state, and X_m the set of marked states. The set of traces *generated* by G is the language

$$L(G) \stackrel{\text{def}}{=} \{s \in \Sigma^*: \delta(x_0, s) \text{ is defined }\},$$

where δ is extended to strings in the usual way. The language $L(G)$ is prefix closed. In this paper, we only deal with prefix-closed languages. Therefore, we assume that $X = X_m$. The notation L/s will be used to denote the post-language of language $L \subseteq \Sigma^*$ after trace $s \in \Sigma^*$:

$$L/s \stackrel{\text{def}}{=} \{t \in \Sigma^*: st \in L\}.$$

In the SSCP framework, a prefix-closed subset of $L(G)$, K, is referred to as the *legal language* (the allowable or desired part of the behavior of G); and, a *supervisor* (or controller) is imposed on G with the task of dynamically enabling/disabling events in order to "achieve" K. The supervisor's task is further complicated by the partitioning of Σ into two subsets: Σ_c, the subset of *controllable* events (those that can be enabled/disabled by the supervisor) and Σ_{uc}, the subset of *uncontrollable* events. Formally, a supervisor or control policy is a function γ: $L(G) \longrightarrow 2^{\Sigma}$, with the following constraint: $(\forall s \in L(G))(\Sigma_{uc} \subseteq \gamma(s))$. The prefix-closed behavior that results from imposing γ on G is denoted by $L(G, \gamma)$ and defined as,

(1) $\epsilon \in L(G, \gamma)$,

(2) $[(s \in L(G, \gamma)) \wedge (s\sigma \in L(G)) \wedge (\sigma \in \gamma(s))] \Leftrightarrow [s\sigma \in L(G, \gamma)]$.

As mentioned above, ideally γ is such that $L(G, \gamma) = K$. In practice, however, such a γ may not exist. Instead, there is a "least restrictive" alternative. This is formally captured by the following definition.

DEFINITION 1 *A prefix-closed sublanguage K is controllable if,*

$$(\forall s \in K, \sigma \in \Sigma_{uc})(s\sigma \in L(G) \Rightarrow s\sigma \in K).$$

Obviously, only controllable subsets (sublanguages) of K can be achieved by means of the control method outlined above. In (Wonham and Ramadge 1987) it is shown that the *supremal controllable sublanguage* of K, the least restrictive solution, denoted by $K^{\uparrow}$, always exists. Moreover, there are algorithms for computing γ's that achieve $K^{\uparrow}$, when $L(G)$ and K are given by *finite* state machines. An on-line algorithm which can provide the control action for a single state at a time (the state just entered by the system), by examining only a portion of $G \times H$ (where $\times$ denotes the Cartesian product and H is a finite state machine generating K) is provided in (Ben Hadj-Alouane et al. 1994c) and reviewed in the next section.

Under the partial observation assumption of (Lin and Wonham 1988), Σ is also partitioned into the subset of observable events, Σ_o, and the subset of unobservable events, Σ_{uo}. The projection, P: $\Sigma^* \longrightarrow \Sigma_o^*$ defined by,

$$P(\epsilon) = \epsilon \quad \text{and} \quad (\forall s \in \Sigma^*, \sigma \in \Sigma)P(s\sigma) = \begin{cases} P(s)\sigma & \text{if } \sigma \in \Sigma_o \\ P(s) & \text{if } \sigma \in \Sigma_{uo}, \end{cases}$$

characterizes the behavior observed by the supervisor, namely $P(L(G))$. The inverse of P is denoted by P^{-1} and is defined as P^{-1}: $2^{\Sigma^*} \longrightarrow 2^{\Sigma^*}$ with $P^{-1}(A) \stackrel{\text{def}}{=} \{t \in \Sigma^*: P(t) = s \text{ for some } s \in A\}$.

A partial observation supervisor is a function γ_p: $P(L(G)) \longrightarrow 2^{\Sigma}$, with the following constraint: $(\forall s \in P(L(G)))(\Sigma_{uc} \subseteq \gamma_p(s))$. The behavior, $L(G, \gamma_p)$, generated by the above supervisor is defined similarly (with $\gamma(s)$ replaced by $\gamma_p(P(s))$. The property of observability, defined below, is key in characterizing the existence of partial observation supervisors.

DEFINITION 2 *A prefix-closed sublanguage K is observable if,*

$$(\forall s, s' \in K, \sigma \in \Sigma)((P(s) = P(s') \wedge s\sigma \in K \wedge s'\sigma \in L(G)) \Rightarrow s'\sigma \in K).$$

Given a prefix closed legal language K, a partial observation supervisor generating K exists if and only if K is controllable and observable (Lin and Wonham 1988). Unfortunately, unlike the full observation case, if K does not satisfy the above condition, there is not an optimal sublanguage that could be generated instead; i.e., the supremal controllable and observable sublanguage need not exist (this is because the union of two observable languages is not necessarily observable). In general, there are many (possibly infinitely many) maximal controllable and observable sublanguages. This paper is concerned with computing them on-line. (Note that for closed languages, the property of observability is

preserved under intersection and hence the infimal closed observable superlanguage exists; see (Lin and Wonham 1988), (Rudie and Wonham 1990).)

The notion of normality (stronger than observability), defined below, is important in partial observation supervision since it results in a supremal normal sublanguage.

DEFINITION 3 *A prefix-closed sublanguage K is normal if,*

$$(\forall s \in L(G))(P(s) \in P(K) \Leftrightarrow s \in K)$$

As it turns out, the supremal controllable and normal sublanguage of the legal language K, denoted by $\sup CN(K)$, also exists and is often offered as a solution to partial observation supervision problems; see, e.g. (Cieslak et al. 1988), (Lin and Wonham 1988) (the computation of $\sup CN(K)$ is addressed in (Brandt et al. 1990), (Cho and Marcus 1989a), (Barbeau et al. 1995)). (One should note however that under the assumption that all the controllable events are observable, the supremal controllable and observable sublanguage exists and coincides with the supremal controllable and normal one.) In this paper, we improve on this situation by specifying the appropriate parameters that force **VLP-PO** to generate maximals that contain the supremal controllable and normal sublanguage.

3. Review of On-Line Control with Full Observation

Variable lookahead control of fully observed systems and with state information was most recently discussed in (Ben Hadj-Alouane et al. 1994c). Under this approach, control actions are determined as needed on-line and by exploring only a portion of the state space of the system using a forward search algorithm. Below, we review from Ben Hadj-Alouane et al. (1994c) only the aspects that are utilized in this paper, namely the prefix-closed version of the algorithm named **VLP-S**. We assume (without loss of generality) that the legal language K is generated by the machine H which is a submachine of G (see Appendix A); thus, we can talk of the legality or illegality of *states* of X rather than *traces* of $L(G)$. The subset of legal states of X is denoted by X_H (which coincides with the state set of H).

The following cost function V (which can be partial) is used to record the relevant information obtained from computing all the previously issued control actions: $V\colon X \longrightarrow \{0, \infty\}$. In the case of prefix closed languages, dealt with here, an infinite cost is assigned to a state if and only if it can uncontrollably lead (i.e., through a trace of uncontrollable events) to an illegal state. The least restrictive full observation policy γ for SSCP is obtained from V as follows:

- If $V(\delta(x_0, s)) = 0$ then $\gamma(s) = \{\sigma \in \Sigma_G(\delta(x_0, s))\colon V(\delta(x_0, s\sigma)) = 0\}$
- If $V(\delta(x_0, s)) = \infty$ then $\gamma(s) = (\Sigma_{uc} \cap \Sigma_G(\delta(x_0, s)))$

where $\Sigma_G(x) \stackrel{\text{def}}{=} \{\sigma \in \Sigma\colon \delta(x, \sigma)$ is defined in $G\}$ denotes the active event set of G at state x.

Each time the system executes a new event, a new control action needs to be determined and issued. For this purpose, values of V which are needed and are yet undefined are computed.

The set $Dom(V)$ denotes the current domain of the partial function V, consisting of those states whose costs have already been computed. The following algorithm **VLP-S** is used to efficiently perform this task for prefix-closed languages (the general algorithm is presented in (Ben Hadj-Alouane et al. 1994c)).

PROCEDURE VLP-S($x_0 \in X$)**:**

1. Exp $:= \emptyset$; Expand(x_0); $D := Dom(V)$;
2. $(\forall y \in ((\text{Exp} - X_H) - D))V(y) := \infty$; $D := Dom(V)$;
3. WHILE $[(\exists y \in \text{Exp} - D)(\exists \sigma \in \Sigma_{uc})(V(\delta(y, \sigma)) = \infty)]$ DO $V(y) := \infty$; $D := Dom(V)$;
4. FOR $[y \in \text{Exp} - D\,]$ DO $V(y) := 0$;
5. RETURN;

Step 1 of **VLP-S** calls the procedure **Expand** which is not shown here for brevity purposes. **Expand** returns in the variable **Exp** the state space of a submachine of G rooted at x. This submachine is expanded in a tree-like manner and includes the states accessible from x that may need to be examined in order to determine the cost of x (for closed languages, this is at most the uncontrollable reach of x, defined as the set of states reachable via an uncontrollable trace from x). In steps 2 and 3, the submachine is trimmed by removing the states that are illegal and states that cannot be prevented (by control) from reaching illegal states.

The **worst-case complexity** of a single execution of **VLP-S** is linear in the number of expanded states. Moreover, the *combined* effort required to determine the cost of *every state* in X is, in the worst case, $O(|X||\Sigma|)$ (this is due to efficient reuse of previous computations; all expanded states are assigned costs and states of known costs are not expanded).

The following lemma and theorem (simplified versions of results given in Ben Hadj-Alouane et al. 1994c) address the correctness of **VLP-S** (the lemma is repeatedly referred to in this paper).

LEMMA 1 $V(x) = \infty \quad \Leftrightarrow \quad K^{\uparrow}/[x] = \emptyset$.

In the above lemma $[x]$ denotes the equivalence class of the state x, i.e., the set of all traces in $L(G)$ that lead to state x from the initial state; $K^{\uparrow}/[x]$ denotes the post-language of $K^{\uparrow}$ from $[x]$. Roughly speaking, the lemma states that when a state is assigned cost ∞ it has an uncontrollable trace (possibly ϵ) emanating from it and leading to an illegal state. The following theorem hints at how **VLP-S** can be used in a procedure to compute a control policy (on-line or off-line) generating the supremal controllable sublanguage of K, $K^{\uparrow}$.

THEOREM 1 $\alpha \in \Sigma \cup \{\epsilon\} \wedge s \in K^{\uparrow} \wedge V(\delta(x_0, s\alpha)) = 0 \quad \Leftrightarrow \quad s\alpha \in K^{\uparrow}$.

4. The VLP-PO Algorithm for On-Line Control under Partial Observation

This section has two subsections. In the first one we give the statement of the algorithm **VLP-PO** and illustrate its operation with an example. In the second subsection we prove

the legality of the control policies computed using **VLP-PO** and determine the worst-case computational complexity of the algorithm.

4.1. Statement of VLP-PO

Under the assumption of full observation, the controller can monitor every event that is executed by the system; hence a new control action can be computed and issued following each event occurrence. In this paper we consider SSCP with the partial observation assumption. Under this assumption, a controller can monitor only a subset of the events, the observable events. As a result, the control action can only be updated following the execution of an observable event; in between two consecutive observable events (while only unobservable events are being executed) the system operates under a constant control action.

In the on-line control paradigm, control actions are determined as needed: Almost instantaneously following the execution of an observable event the next control action is computed and issued. In this paper we only deal with closed languages (i.e., blocking is not an issue). Hence, intuitively, determining the next control action involves ensuring that the execution of any trace consisting of a sequence of unobservable events possibly followed by a single observable event (the new control action remains in effect until the next observable event is executed) does not cause the system to generate illegal behavior, either directly or via an uncontrollable trace. To state this more precisely, we need to define the following operators.

- The **unobservable reach** of the subset of states $S \subseteq X$ under the subset of events $E \subseteq \Sigma$ is given by,

$$\mathrm{UR_E}(S) \stackrel{\text{def}}{=} \{x \in X\colon\ x = \delta(y, u) \text{ for } y \in S \text{ and } u \in (\Sigma_{uo} \cap E)^*\}.$$

  $\mathrm{UR_E}(S)$ consists of all the states reachable from states in S via traces of unobservable events from E.

- The **extended unobservable reach** of the subset of states $S \subseteq X$, under the subset of events $E \subseteq \Sigma$ is given by,

$$\mathrm{UR{+}_E}(S) \stackrel{\text{def}}{=} \mathrm{U}R_{\mathrm{E}}(S) \ \cup\ \{x \in X\colon\ x = \delta(y, \beta), \quad \text{for } y \in \mathrm{UR_E}(S) \text{ and } \beta \in (\Sigma_o \cap E)\}.$$

  $\mathrm{UR{+}_E}(S)$ consists of all the states reachable from states in S via unobservable traces over E or unobservable traces over E followed by a single observable event in E.

If S denotes the set of states that the system could be in immediately following the execution of the last observed event, then a valid next control action E must be such that $\mathrm{UR{+}_E}(S)$ does not contain any "bad" states (the **VLP-S** algorithm presented in Section 3 associates an infinite cost with bad states). Since it is well know that in the case of SSCP with partial observation the supremal controllable and observable sublanguage of the given legal language does not always exist, different strategies for determining a valid next control action may exist (there is no "optimal" next control action).

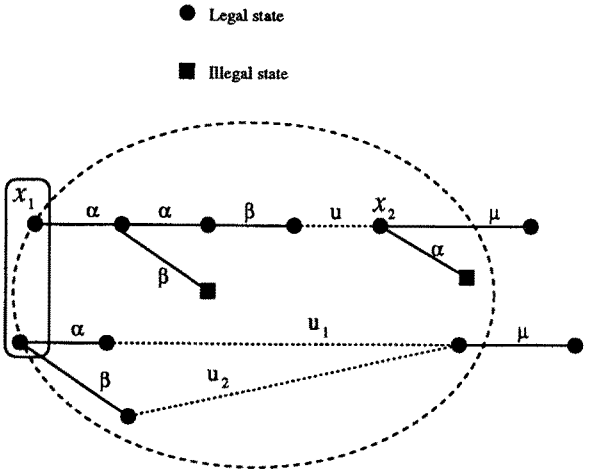

Figure 1. Unobservable reach.

In (Heymann and Lin 1994) such a strategy is presented [1]. Its starting point is a precomputed full observation least restrictive control policy [2] for the SSCP problem at hand. Given S as specified above (i.e., the possible states just prior to application of the next control action), each state in $\mathrm{UR}_\Sigma(S)$, the largest possible unobservable reach of S, is examined. An event is excluded from the next control action E if and only if it is disabled by the full observation control policy at a given state in $\mathrm{UR}_\Sigma(S)$; i.e.,

$$E = \bigcap_{x \in \mathrm{UR}_\Sigma(S)} \gamma(x),$$

where γ denotes the least restrictive full observation control policy. From the definition of γ given in Section 3 (using the terminology of the **VLP-S** algorithm), an event is disabled at a given state if and only if it leads to a bad (infinite cost) state; it is easy to see how the extended unobservable reach of S resulting from the above described control action of (Heymann and Lin 1994) consists entirely of good (zero cost) states.

The above strategy, though simple, is in a "local sense" too restrictive. To illustrate this, consider the situation depicted in Figure 1. The states in S are enclosed by a rectangle; the states in $\mathrm{UR}_\Sigma(S)$ are enclosed by a dashed ellipse (though not all the states of $\mathrm{UR}_\Sigma(S)$ are displayed, all the traces leading to a displayed state are shown). Disabling the unobservable β, for example, automatically excludes the state x_2 (reachable only via $\alpha\alpha\beta u$ from x_1 in S) from the unobservable reach generated by the next control action. However, the strategy of (Heymann and Lin 1994) would still examine x_2 and as a result the event α is disabled (α is not in $\gamma(x_2)$ since $\delta(x_2, \alpha)$ is an illegal state).

The immediate "remedy" to the above situation requires incrementally growing the current unobservable reach as more events are added to the next control action (it should be pointed out that while a local improvement of the behavior is guaranteed, a global improvement may not necessarily follow; this is shown in the next section). However, in committing to the

above process, it is important to be aware of the following issue. Mutually exclusive choices of events may be involved in the construction of the next control action; and this is one of the reasons why a supremal controllable and observable sublanguage does not always exist. This situation is illustrated again by Figure 1: Ensuring that $\mathrm{UR}_E(S)$ (where E denotes the next control action) consists entirely of good states could be achieved by either disabling α or β. More importantly, the resulting behaviors (i.e., the one resulting from a control action that disables β and the one resulting from a control action that disables α instead) are incomparable with respect to set inclusion (notice that the first behavior excludes the continuation of S, β, which is included by the second behavior).

Our on-line approach is captured by the algorithm **VLP-PO** which is given below. **VLP-PO** uses a new strategy for computing the next control action, based on the observations outlined above. (The initialization of **VLP-PO** is discussed later in this section.)

PROCEDURE **VLP-PO**($\beta_o \in \Sigma_o \cup \{\epsilon\}$);

1. NS := N_{β_o}(PS);
2. **Control-Action(NS, Event-Ordering)**;
3. PS := $\mathrm{UR}_{\mathrm{ACT}}$(NS);
4. END;

Immediately following the execution of a new observable event β_o, the call **VLP-PO**(β_o) is executed. When this execution terminates the next control action is found in the the global variable ACT. Another global variable, PS, is used by **VLP-PO** to store the set of *possible states* the system could be in while executing any unobservable event under the newly computed control action (this is the set of possible states immediately after issuing the returned control action and prior to the execution of the next observable event). The value of PS computed by the last call to **VLP-PO** is used in Step 1 to compute NS, the **next** set of states the system could be in immediately following the execution of the last observed event (this is the set of possible states just prior to the application of the next control action); NS is given by the operator N_σ defined below. For $S \subseteq X$ and $\sigma \subseteq \Sigma \cup \{\epsilon\}$,

$$\mathrm{N}_\sigma(S) \stackrel{\text{def}}{=} \{x \in X\colon\ x = \delta(y, \sigma) \text{ for } y \in S\}.$$

NS is used in Step 2 by the procedure **Control-Action**, given below, to determine the next control action. The above process is illustrated by Figure 2.

PROCEDURE **Control-Action**($S \subseteq X$, **EList : Ordered-**Σ_c);

1. ACT := Σ_{uc}; Pt := 1;
2. WHILE [EList $\neq \emptyset$] DO
- 2.1 IF [Pt > | EList |)] THEN
  ACT := ACT $\cup$ EList; RETURN;
- 2.2 IF [$(\forall x \in \mathrm{UR}_{+\mathrm{ACT}}(S))(\delta(x,$ EList.Pt) is not defined)] THEN
  Pt := Pt +1; GOTO 2;

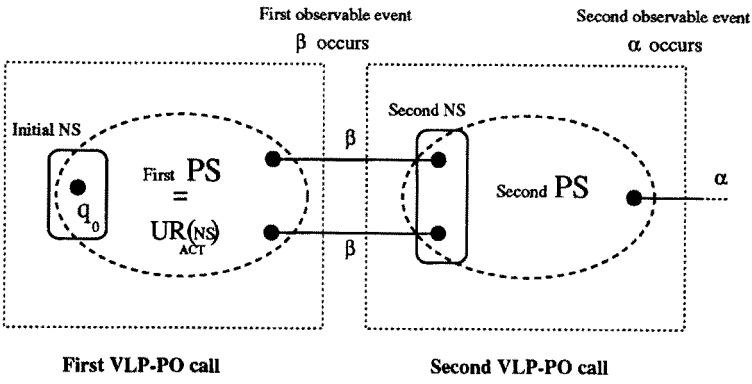

Figure 2. VLP-PO operation.

2.3 FOR [$x \in \mathrm{UR} +_{\mathrm{ACT} \cup \{\mathrm{EList.Pt}\}} (S)$] DO
IF [$x \notin \mathrm{DOM}(V)$] THEN VLP-S(x);
IF [$\mathrm{V}(x) = \infty$] THEN
EList := EList - { EList.Pt }; GOTO 2;

2.4 ACT := ACT ∪ { EList.Pt }; EList := EList − { EList.Pt }; Pt := 1; GOTO 2;

3. END;

Our strategy for computing the next control action is best described as completely on-line and priority-based. It is completely on-line in the sense that the control action and its corresponding extended unobservable reach ($\mathrm{UR} +_{\mathrm{ACT}} (S)$) are constructed incrementally, and hence states are never unnecessarily examined. Moreover, state costs are computed on-line, as needed, using the algorithm **VLP-S** given in Section 1. The strategy is priority-based in the sense that the control action is constructed from a given total ordering of the controllable events (stored in the global variable Event-Ordering before a call is issued to **VLP-PO**). This ordering can be changed from one control action computation to the next (a different ordering may result in a different control action). This changing of the priorities can be a useful tool in "steering" the resulting system behavior (see Example 1 and Section 6).

The specifics of operation of the procedure **Control-Action** are as follows. In Step 1, ACT is initialized to the set of uncontrollable events; by definition, these events cannot be disabled. The WHILE LOOP in Step 2 examines the controllable events in the order specified by EList (which contains the ordering discussed above). The event currently being examined is pointed to by Pt. At most two tests can be conducted in relation to the event being examined. Step 2.2 determines if the examined event can add to the already enabled behavior; i.e., whether or not $\mathrm{UR} +_{\mathrm{ACT}} (S) \neq \mathrm{UR} +_{\mathrm{ACT} \cup \{\mathrm{Elist.Pt}\}} (S)$. If the event does not add to the current behavior then it remains on the list and the next event on the list is examined (i.e., back to Step 2). Step 2.3 determines if the extended unobservable reach resulting from enabling the event (in addition to the events that are already enabled) has any

bad (infinite cost) states: **VLP-S** is called to determine the costs of any newly encountered states. Following this test the examined event is removed from Elist (it will not be further examined by the WHILE LOOP); it is *permanently included* in the next control action if and only if it does not incur any bad states in the resulting extended unobservable reach (this is a part of Step 2.3). If in Step 2.1 Pt points past the size of Elist the procedure **Control-Action** terminates. There are two cases where this can happen. In the first case Elist is empty; hence a decision has been taken with respect to each controllable event and the task of determining the next control action is complete. In the second case, none of the events currently on Elist add to the current behavior; hence it makes no difference (from a behavior viewpoint) whether the events on Elist are enabled or disabled, but we choose to enable them. The test of Step 2.3 is required to ensure the validity of the next control action. As we shall prove later, Step 2.2 is necessary to guarantee the maximality of the resulting behavior. The event priorities provide a natural solution to the situation of mutually exclusive events discussed earlier.

The following conditions are needed for the proper operation of **VLP-PO**, including its initialization (some of them are discussed above).

- **PS** $:= \{x_0\}$ prior to the first call to **VLP-PO**, where x_0 is the initial sate of G.
- **Event-Ordering** := a total ordering on Σ_c. This ordering need not be the same for each control action computation.
- **VLP-PO**(ϵ) must be used to determine the first control action.

The following example illustrates the operation of **VLP-PO**.

EXAMPLE 1 Consider the finite state machine in Figure 3. Assume that all the events are controllable. Moreover, the events labeled with α are observable while the ones labeled with β are unobservable.

PS is initialized to the singleton containing the initial state and the call **VLP-PO**(ϵ) is issued to compute the first control action. The current set of next states (the value of NS following Step 1 of **VLP-PO**) is enclosed in the small rectangle; it includes only the initial state. The left-most ellipse encloses the largest possible unobservable reach of NS: $UR_\Sigma(NS)$.

Consider the following ordering of controllable events: The β's are placed first, in accordance with their subscripts, followed by the α's, also in accordance with their subscripts. With this ordering the event β_1 is the first to be enabled. When examining β_2 the test in Step 2.2 fails (enabling the event does not add a new trace to the current extended unobservable reach consisting only of the trace β_1) and the event is skipped. The event β_3 is the second to be enabled. When β_2 is re-examined it passes the test of Step 2.2 but fails test of Step 2.3 (it extends the trace β_3 to $\beta_3\beta_2\beta_3$, which leads to an illegal state) and it is disabled. The events α_1 and α_2 are the last to be enabled. Hence, the first control action is $\{\beta_1, \beta_3, \alpha_1, \alpha_2\}$. The value of PS resulting from this control consists of the initial state and the two states reachable from the initial state via the traces β_1 and β_3. Note that the above control action cannot be improved upon; in fact it is the best possible control action (optimal in a local sense).

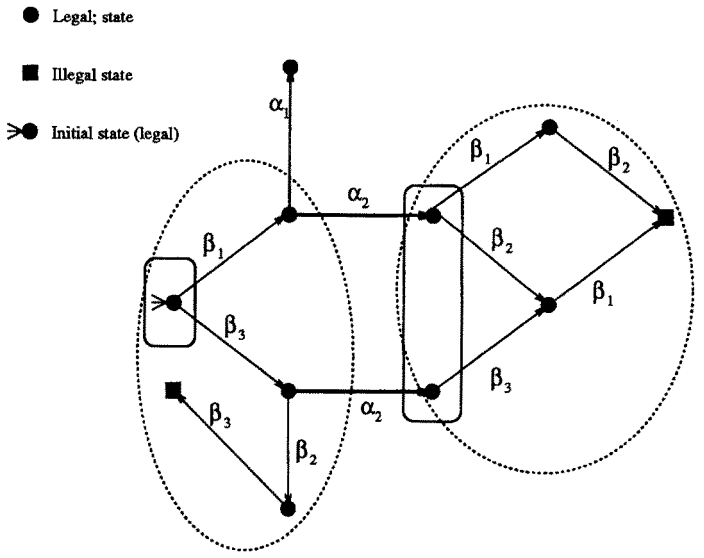

Figure 3. System illustrating VLP-PO execution.

Let the next observable event executed by the system be α_2. Then the call **VLP-PO**(α_2) must be issued to compute the second control action. The current NS is enclosed by the large rectangle; it consist of all the successors of the states in PS (given above) via the event α_2. By maintaining the previous ordering, we get the following control action $\{\beta_1, \alpha_1, \alpha_2\}$. Using an ordering which differs from the above by simply interchanging the positions of β_1 and β_2 we get a different control action, namely $\{\beta_2, \beta_3, \alpha_1, \alpha_2\}$. Note that neither control action can be improved upon (the resulting behaviors are incomparable with respect to set inclusion).

4.2. Correctness and Complexity

By the definition of a partial observation control policy (see Section 2), all the uncontrollable events must be part of any control action (that is why Step 1 of **Control-Action** initializes ACT to Σ_{uc}). As a result, a control policy γ_p generating a non-empty subset of K may not always exist: Sometimes illegal behavior cannot be avoided even at the onset of system operation (i.e., at the initial state), even by disabling all the controllable events. In such a situation a starting error – SE – is said to occur (Chung et al. 1992) (the system cannot be safely *started*, or it will generate an erroneous behavior no matter how it is started). The notion of an SE is formalized below.

DEFINITION 4 *A system G is said to incur an* **SE** *with respect to a given legal language K if there does not exist a control policy γ_p such that $L(G, \gamma_p) \subseteq K$.*

The following proposition provides a simple test for identifying an SE situation (it basically formalizes what is stated above).

PROPOSITION 2 *There is no SE if and only if* $K^{\uparrow}$ *is non-empty.*

Proof: Assume that there is no SE. Then there must exist some γ_p such that $L(G, \gamma_p) \subseteq K$. Consider the simple policy $\gamma_p{}'$ such that $(\forall s \in P(L(G)))\gamma_p{}'(s) = \Sigma_{uc}$: Obviously, $\gamma_p{}'$ is a sub-policy of γ_p (i.e., it is always more constraining); hence $L(G, \gamma_p{}') \subseteq K$. By definition $L(G, \gamma_p{}')$ is controllable and non-empty, since it always contains the empty trace; hence, $K^{\uparrow} \neq \emptyset$.

Assume $K^{\uparrow} \neq \emptyset$. Then there must exist a full observation policy γ such that $L(G, \gamma) = K^{\uparrow}$. Furthermore, the above policy $\gamma_p{}'$ when viewed as a full observation policy (i.e., all strings of the same equivalence class are assigned the same control action), is a sub-policy of γ. Hence $L(G, \gamma_p{}') \subseteq K$, and there is no SE. ■

The following result states that **VLP-PO** is correct.

THEOREM 2 *If* $K^{\uparrow} \neq \emptyset$, *then for each* γ_p *computed using* **VLP-PO**, $L(G, \gamma_p)$ *is a controllable and observable sublanguage of* K.

Note that the precondition of the above result, $K^{\uparrow} \neq \emptyset$, is not a limitation of the algorithm; as explained above, it is an indication of an SE situation, where essentially nothing can be done from a control viewpoint. The proof of the theorem is given in Appendix B.

The last result of this section gives the worst-case computational complexity of each call to **VLP-PO** in terms of $|X|$, $|\Sigma|$ and $|\Sigma_c|$, respectively, the cardinalities of the state space of G, of the event set of G and of the subset of controllable events.

THEOREM 3 *In the worst case, the effort involved in computing a control action using* **VLP-PO** *is* $O(\max(|X||\Sigma_c|^2, |X||\Sigma|))$.

Proof: Clearly, each of steps 1 and 3 of **VLP-PO** requires at most $O(|X||\Sigma|)$ to execute. The effort required by step 2, consisting of the execution of a call to **Control-Action**, is tabulated as follows.

Step 1 of **Control-Action** requires no more than $O(1)$. The requirements of the WHILE LOOP, at Step 2, are determined below.

First we show that the loop is executed at most $|\Sigma_c|^2$ times. Following each iteration, the examined event, EList.Pt, is either removed (Step 2.4), in which case the event pointer, Pt, is reset to the top of EList, or Pt is advanced; the loop terminates whenever the event pointer is advanced beyond the the size of the list (Step 2.1). If the current size of Elist is n then in the worst case, Pt is advanced until the position of the nth event (without removing any events from EList) and upon examining the nth event, it is either removed, in which case the size of EList becomes $n-1$ and the process repeats, or the loop terminates. Given that the original size of EList is $|\Sigma_c|$, the loop cannot execute more than $(|\Sigma_c|+|\Sigma_c|-1+|\Sigma_c|-2+\cdots+1) \leq |\Sigma_c|^2$ times.

Each of steps 2.1 and 2.2 require no more than $O(|X|)$. Step 2.3 involves, in the worst case, examining all the states in X and for each examined state either work requiring $O(1)$ is performed or, in addition to that, **VLP-S** is invoked to compute an unknown state cost.

From Section 3, the total added effort required by **VLP-S** to compute the cost of all states in X is $O(|X||\Sigma|)$. Hence, the total effort expanded on **VLP-S** calls during the entire WHILE LOOP execution is no more than $O(|X||\Sigma|)$. The remaining effort required by Step 2.3 is at most $O(|X|)$ per iteration. Hence the total effort required by the WHILE LOOP execution is $O(|X||\Sigma_c|^2) + O(|X||\Sigma|) = O(\max(|X||\Sigma_c|^2, |X||\Sigma|))$.

Therefore an execution of **Control-Action** requires no more than

$$O(\max(|X||\Sigma_c|^2, |X||\Sigma|)).$$

When the efforts required by the individual steps of **VLP-PO** are added, it is clear that the total effort required by a single execution of **VLP-PO** is no greater than $O(\max(|X||\Sigma_c|^2, |X||\Sigma|))$. ■

5. Properties of VLP-PO Policies

We first present an important result concerning the quality of the policies computed using **VLP-PO**, namely that they are guaranteed to generate maximal controllable and observable sublanguages of K. Then a separate subsection presents a result highlighting an interesting feature of **VLP-PO**: By proper parameter selection, the resulting policies are guaranteed to contain $\sup CN(K)$, the supremal controllable and normal sublanguage of K. We also discuss some interesting problems that cannot be addressed using **VLP-PO**; counter-examples are presented to support these discussions.

5.1. *Maximality*

The following theorem shows the maximality of the language generated by the **VLO-PO** algorithm.

THEOREM 4 *Let γ_p be a policy computed using* **VLP-PO**. *If $K^{\uparrow} \neq \emptyset$, then $L(G, \gamma_p)$ is a maximal controllable and observable sublanguage of K.*

Proof: (by contradiction): Assume that $L(G, \gamma_p)$ is *not* a maximal controllable and observable sublanguage of K. By Theorem 2, $L(G, \gamma_p)$ is an controllable and observable sublanguage of K; hence, there must exist a partial observation policy $\gamma_{p'}$ such that $L(G, \gamma_{p'})$ is an controllable and observable sublanguage of K and $L(G, \gamma_p) \subset L(G, \gamma_{p'})$. This implies that there must exist $s \in P(L(G, \gamma_p))$ such that,

$$(P^{-1}(s) \cup P^{-1}(s)\Sigma_o) \cap L(G, \gamma_p) \subset (P^{-1}(s) \cup P^{-1}(s)\Sigma_o) \cap L(G, \gamma'_p) \tag{1}$$

$$(\forall t < s) P^{-1}(t) \cap L(G, \gamma_p) = P^{-1}(t) \cap L(G, \gamma'_p). \tag{2}$$

Equation (1) follows from the fact that since $L(G, \gamma_p) \subset L(G, \gamma_{p'})$, there has to be a point where $\gamma_{p'}$ allows more events than γ_p. Let that occur at trace s and look at all that $L(G, \gamma_p)$ and $L(G, \gamma_{p'})$ can do after s, until the next control action is issued; then we see that equation

(1) holds. If we pick the first such s, then equation (2) is true. Note that if $s = v\sigma$ (i.e., $s \neq \epsilon$), we have that (from (2) and the fact that $s \in P(L(G, \gamma_p))$)

$$(P^{-1}(v)\sigma) \cap L(G, \gamma_p) = (P^{-1}(v)\sigma) \cap L(G, \gamma_p'). \tag{3}$$

(If equation (3) were not true, this would contradict the observability of $L(G, \gamma)$ proved previously.) Hence from Equation (1), there must exist a $\beta \in \Sigma_c$ such that $\beta \in \gamma_p'(s)$ and $\beta \notin \gamma_p(s)$.

Consider the computation of $\gamma_p(s)$ by **VLP-PO** (i.e., the computation of ACT immediately after observing s). By an induction argument similar to the one used in the proof of Theorem 2 and by the initial conditions of **VLP-PO** it can be shown that,

$$\begin{aligned} \mathrm{NS}_s &= \{x_0\} \\ &\quad \text{for } s = \epsilon \end{aligned} \tag{4}$$

$$\begin{aligned} \mathrm{NS}_s &= \{x \in X\colon x = \delta(x_0, t) \text{ for } t \in (P^{-1}(v)\sigma) \cap L(G, \gamma_p)\} \\ &\quad \text{for } s = v\sigma \text{ with } \sigma \in \Sigma_o. \end{aligned} \tag{5}$$

Moreover, at the the point of the computation when β is disabled we must have by Step 2.3 of **Control-Action** that $(\exists x \in \mathrm{UR} +_{\mathrm{ACT}\cup\{\beta\}} (\mathrm{NS}_s))V(x) = \infty$. Also, by Step 2.2 of **Control-Action**, every event that is already in ACT contributes to the behavior (a crucial fact for this proof). It follows from equations (3), (5) and (1) that $\mathrm{ACT}\cup\{\beta\} \subseteq \gamma_p'(s)$ and that $(\exists t \in L(G, \gamma_p'))\delta(x_0, t) = x$. Therefore, from Lemma 1 and the fact that K is prefix-closed, we have that $(\exists u \in \Sigma_{uc}^*)\delta(x_0, tu) \notin K$. By the controllability of $L(G, \gamma_p\prime)$ it follows that $tu \in L(G, \gamma_p')$ and, hence, $L(G, \gamma_p') \not\subseteq K$. This clearly contradicts our proof hypothesis. Hence, $L(G, \gamma_p)$ must be a maximal controllable and observable sublanguage of K. ■

The following corollary highlights an interesting feature of the previous result.

COROLLARY 3 *If $K^{\uparrow} \neq \emptyset$ and if the* **supremal** *controllable and observable sublanguage of K exists, then this supremal is generated by every policy computed using* **VLP-PO**.

It was shown in the previous section (see Example 1) how different maximal controllable and observable sublanguages can be generated by varying the controllable events ordering given to **VLP-PO** each time it is called. The following example shows, however, that not all maximal controllable and observable sublanguages can be generated using **VLP-PO** policies. (This example contradicts the claim incorrectly made in Theorem 7 of (Ben Hadj-Alouane et al. 1993).)

EXAMPLE 2 Consider the finite state machine in Figure 4. Assume that all the events are controllable and that only the event o is observable.

The first call to **VLP-PO** (with NS consisting only of the initial state) results in $\gamma_p(\epsilon) = \{\beta, \alpha, o\}$ irrespective of the ordering of controllable events used. The second call to **VLP-PO** is issued after the event o is executed; it uses the NS enclosed by the right-most

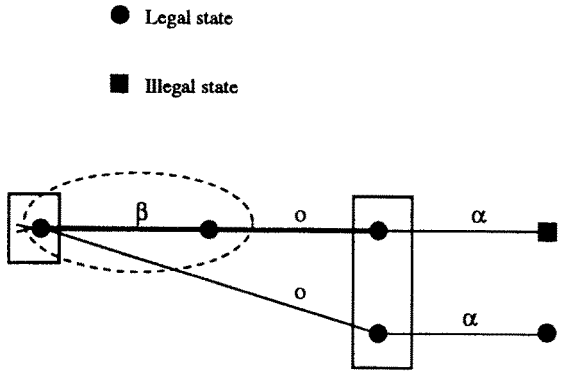

Figure 4. Maximal not achievable.

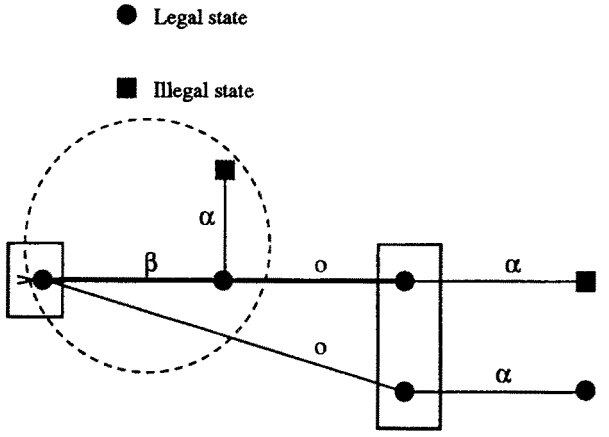

Figure 5. Other maximal not achievable.

rectangle. Hence, α must be disabled. The resulting language generated by the only possible **VLP-PO** policy is $\overline{\{o, \beta o\}}$, where - denotes the prefix closure. Note, however, that $\overline{\{o\alpha\}}$ is another maximal controllable and observable sublanguage of the legal language; generating it requires disabling β at the first control action, something that runs counter to the local optimization that **VLP-PO** performs.

The first control action required to generate the maximal sublanguage $\overline{\{o\alpha\}}$ (i.e., $\{\alpha, o\}$) does not constitute a "maximal control action"; i.e., as specified above, $\{\beta, \alpha, o\}$, which is a superset of the above control action, also constitutes a legal first control action. In fact one can talk about a maximal control policy (where each control action constitutes a maximal among the set of legal control actions), and clearly the above maximal sublanguage is not generated by a maximal control policy. It can be easily verified that all sublanguages computed using **VLP-PO** are generated by maximal control actions. However, not all maximal sublanguages with maximal control policies can be computed using **VLP-PO**.

This is demonstrated by the finite state machine of Figure 5 (a modified version of the finite state machine of Figure 4). Note that $\overline{\{o\alpha\}}$ is a maximal sublanguage that can be generated with the maximal control policy $\gamma_p(\epsilon) = \{\alpha, o\}$ and $\gamma_p(o) = \{\beta, \alpha, o\}$. However, this sublanguage cannot be computed with **VLP-PO**, because, in the first control action, α cannot be enabled without first enabling β (α cannot generate new behavior with β disabled).

5.2. *Maximals Containing the Supremal Controllable and Normal*

The following result shows that by an appropriate selection of the ordering of the controllable events given to **VLP-PO** each time it is called, the maximal controllable and observable sublanguage that is generated is guaranteed to contain $\sup CN(K)$. Moreover, this ordering can be *statically determined*, in the sense that it is both independent of K and of the currently observed trace for which a control action is to be computed.

THEOREM 5 *If γ_p is a policy computed with* **VLP-PO** *using controllable events orderings placing all the unobservable events first, then* $\sup CN(K) \subseteq L(G, \gamma_p)$.

Proof: (by contradiction): Assume that $\sup CN(K) \not\subseteq L(G, \gamma_p)$ and take the shortest trace that is in $\sup CN(K)$ and not in $L(G, \gamma_p)$; i.e., we have $s \in \sup CN(K) \cap L(G, \gamma_p)$ and $s\sigma \in \sup CN(K) - L(G, \gamma_p)$, for $\sigma \in \Sigma$. Clearly, $\sigma \in \Sigma_c$ and $\sigma \notin \gamma_p(P(s))$.

Consider the computation of $\gamma_p(P(s))$ by **VLP-PO** and more specifically the point at which σ was disabled: From Step 2.3 of **Control-Action** it follows that $(\exists x \in \text{UR} +_{\text{ACT}\cup\{\sigma\}} (\text{NS}))V(x) = \infty$. There are exactly two cases.

Case $\sigma \in \Sigma_{uo}$: We must have $\text{ACT} \cap \Sigma_c \subseteq \Sigma_{uo}$. To see this, observe that from the hypothesis of the theorem (concerning the ordering of events), each event in $\Sigma_c \cap \Sigma_{uo}$ must be examined at least once by **Control-Action** before any observable event is examined. As a result, by the time the first observable event is examined, the unobservable events that are on Elist are surely to remain on EList until the WHILE LOOP terminates and are added to ACT by Step 2.1, since they can never cause Step 2.3 to execute (adding observable events to ACT will not help unobservable events generate new behavior in $\text{UR} +_{\text{ACT}} (\text{NS}_s)$).

Hence, x is reachable from a state y in NS either via $t \in \Sigma_{uo}^+$ or via a $r\beta$, where $r \in \Sigma_{uo}^*$ and $\beta \in \Sigma_{uc}$; moreover, in the second case, we have $V(\delta(y, r)) = \infty$ (a simple application of Lemma 1). Let $\delta(x_0, w) = y$; clearly, $wt \notin K^\uparrow$ and $wr \notin K^\uparrow$ (simple applications of Theorem 1). However, $wr \in P^{-1}(s)$ and $wt \in P^{-1}(s)$; hence, $wt \in \sup CN(K)$ and $wr \in \sup CN(K)$, by the definition of normality. This contradicts the fact that $\sup CN(K) \subseteq K^\uparrow$.

Case $\sigma \in \Sigma_o$: It must be that x is reachable from some $y \in \text{NS}$ via $r\sigma$, for $r \in \Sigma_{uo}^*$ (adding the *observable* event σ to the current ACT can add new states to the extended unobservable reach; however, all the new states are successors via σ of states already in the unobservable reach). Let $\delta(x_0, w) = y$; with a simple application of Theorem 1 we can see that $wr\sigma \notin K^\uparrow$. At the same time, we have $wr\sigma \in P^{-1}(s\sigma)$, and $s\sigma \in \sup CN(K)$. By normality, $wr\sigma \in \sup CN(K)$. This again contradicts the fact that $\sup CN(K) \subseteq K^\uparrow$.

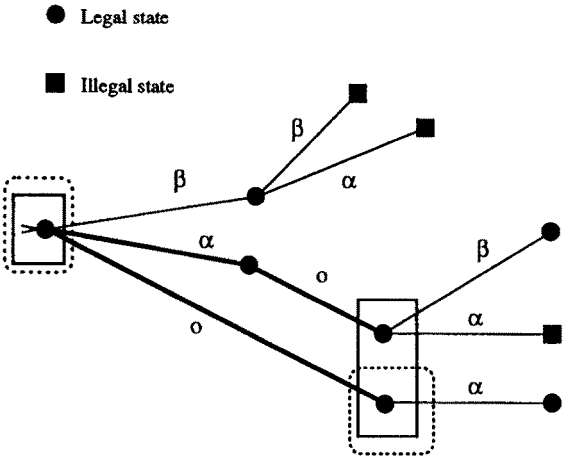

Figure 6. Incomparable policies.

Both of the above cases lead to a contradiction, and therefore $\sup CN(K) \subseteq L(G, \gamma_p)$. ■

In (Ben Hadj-Alouane et al. 1993) a different method, based on a dynamically computed ordering scheme, is presented to generate maximal controllable and observable sublanguages that contain $\sup CN(K)$ using **VLP-PO**. For each new control action, a suitable ordering of controllable events is first computed based on the control action given by the strategy of (Heymann and Lin 1994) described in Section 4 (the events enabled by this control action are placed first in the ordering); this ordering is then used to call **VLP-PO** in the usual way to compute the real control action. It can be easily shown (though we do not pursue it for brevity purposes) that $L(G, \gamma_p)$ obtained in this way includes $L(G, \gamma_\pi)$, where γ_π is the first policy of (Heymann and Lin 1994) (two policies are described in that paper, and the second one subsumes the first) [3]. In (Heymann and Lin 1994) it is shown that $L(G, \gamma_\pi)$ contains $\sup CN(K)$. Hence, $L(G, \gamma_p) \supseteq \sup CN(K)$.

The above results give two examples of sublanguages, $\sup CN(K)$ and $L(G, \gamma_\pi)$, that can be "maximized" using **VLP-PO**. However, the following example shows that **VLP-PO** cannot be used to compute a maximal controllable and observable sublanguage containing $L(G, \psi_\pi)$, where ψ_π is the second policy presented in (Heymann and Lin 1994).

EXAMPLE 3 Consider the finite state machine in Figure 6. Assume that all the events are controllable and that o is the only observable event.

The first call to **VLP-PO** (with NS consisting only of the initial state) results in $\gamma_p(\epsilon) = \{\alpha, o\}$ irrespective of the controllable events ordering used. The second call to **VLP-PO** issued after the event o is executed uses the NS enclosed by the right-most solid rectangle. Hence, α must be disabled in the second control action. The resulting language generated by the only possible **VLP-PO** policy is $\overline{\{o, \alpha o \beta\}}$.

The first control action of the second policy of (Heymann and Lin 1994), ψ_π (computed

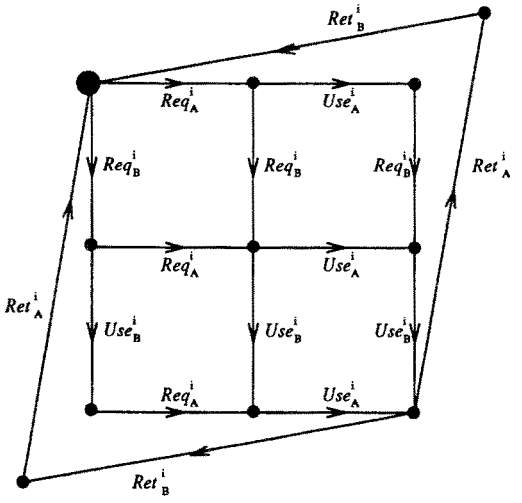

Figure 7. Model of user i.

as described in Section 4), enables only the event o. The NS that it uses to compute its second control action, after observing o, is enclosed by the right-most dotted rectangle; as a result, the second control action enables all events. The language generated by this policy is $\overline{\{o\alpha\}}$ which is incomparable (with respect to set inclusion) to the one generated by the above **VLP-PO** policy.

6. A Resource Allocation Problem

In this section we present a more detailed illustrative example of the use of the **VLP-PO** algorithm. The purpose of this example is to compare **VLP-PO** with (i) the control policy implementing the supremal controllable normal sublanguage and (ii) the second policy of (Heymann and Lin 1994). We also use this example to comment on typical problems encountered in attempting to tackle partial observation problems with currently available synthesis tools (including the one presented in this paper).

We consider a resource allocation problem with one resource pool dispensing two resources, denoted by A and B, to two users numbered 1 and 2. We assume that the users communicate directly with the resource pool to ensure mutual exclusion; i.e., a user issues a request for a particular resource to the pool, and only when permitted by the pool can the user access the resource. Also, the resource is returned back to the pool when the task involving the resource is completed.

The models of user i, denoted by G_{ui}, and the resource pool, denoted by G_p, are show in Figures 7 and 8 (in each figure, the initial state is the bold one). The events Req^i_A, Use^i_A and Ret^i_A, capture, respectively, a request for resource A by user i, the granting of permission by the resource pool to user i to access resource A, and the return of resource A by user i to the resource pool. The complete system model G consists of the parallel (also called

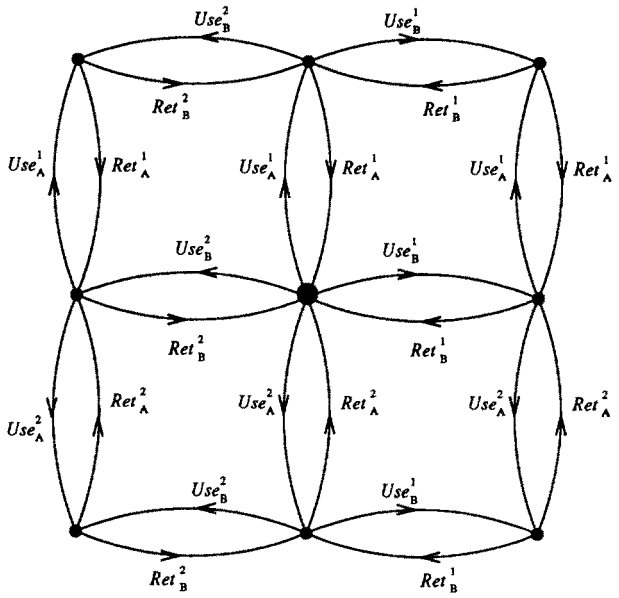

Figure 8. Model of resource pool.

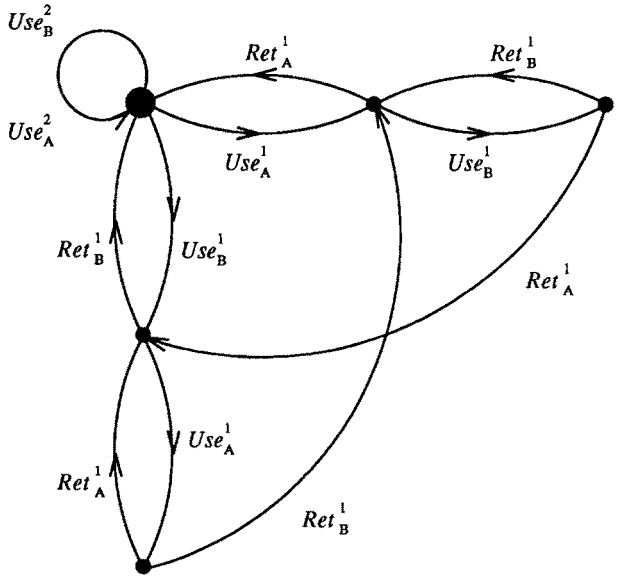

Figure 9. Non-blocking specification for user 1.

synchronous) composition of the three models G_{u1}, G_{u2} and G_p, i.e., $G = G_{u1}||G_{u2}||G_p$.

We assume that each user will request and return a resource in finite time. Each user

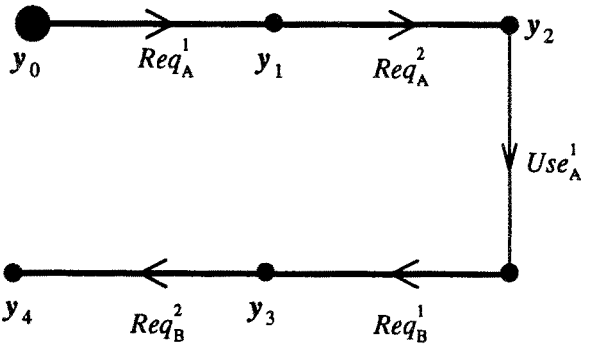

Figure 10. Blocking solution.

must acquire both resources (in any order), presumably to perform an unmodeled task (for simplicity purposes), before returning them to the pool. To avoid a "deadlock" situation where a given user is allowed to acquire any particular resource and, at the same time, the other user is allowed to acquire the other resource, we impose a legal specification K_1 shown in Figure 9 on user 1 which disallows the above situation. A similar specification K_2 for user 2 can be obtained from the specification K^1 by replacing each event with the superscript 1 by the comparable event with a superscript 2 and vice versa. A model of the complete specification K_s is obtained by taking the parallel composition of the above two specifications, i.e., $K_s = K^1 || K^2$. The desired legal language is $K = K_s \cap L(G)$.

Consider the supervision of the system G for the purpose of achieving a subset of the legal language K with $\Sigma_c = \{\text{Use}_A^i, \text{Use}_B^i\colon\ i = 1, 2\}$ and with $\Sigma_o = \{\text{Req}_A^i, \text{Req}_B^i, \text{Ret}_A^i, \text{Ret}_B^i\colon\ i = 1, 2\}$. (Observe that $K^{\uparrow} \neq \emptyset$ for this choice of controllable events.) The application of **VLP-PO** requires transforming the machines of G and K_s using the procedure described in Appendix A (so that K is generated by a submachine of the machine generating $L(G)$); note that the actual construction of G and K_s (as parallel compositions) and the application of this procedure can be performed on-line, as needed by **VLP-PO** (a new state is constructed only if it is part of the current extended unobservable reach or if it is required by **VLP-S** to compute the cost of a state in the current extended unobservable reach). The path shown in Figure 10 can be executed by the system under the **VLP-PO** policy partially given in Table 1 (as it turns out the ordering is irrelevant in computing the control actions of Table 1). Note that after reaching y_4 the system cannot execute any additional events *under the given control action* and "blocks". Since the control policy enabled both users to access both resources at y_0 and since the Use events are unobservable, access to resource B must be denied to both users from y_3 and forward (accounting for this situation requires global optimization [4], which is likely to be computationally expensive, if systematically feasible). This problem can be solved by making the Use events observable (possibly instead of the Req events). We note here that the second policy of (Heymann and Lin 1994) also generates the path of Figure 10 and that $\sup CN(K) = \emptyset$.

Next, for the sake of illustration, we briefly discuss the case where Σ_o is modified to $\Sigma_o = \{\text{Ret}_A^i, \text{Ret}_B^i\colon\ i = 1, 2\}$. In this case, the path shown in Figure 11 can be executed by

Table 1. Policy for blocking solution.

State	Control Action
y_0	$\{Use_A^i, Use_B^i\}$
y_1	$\{Use_A^i, Use_B^i\}$
y_2	$\{Use_A^i, Use_B^i\}$
y_3	$\{Use_A^i\}$
y_4	$\{Use_A^i\}$

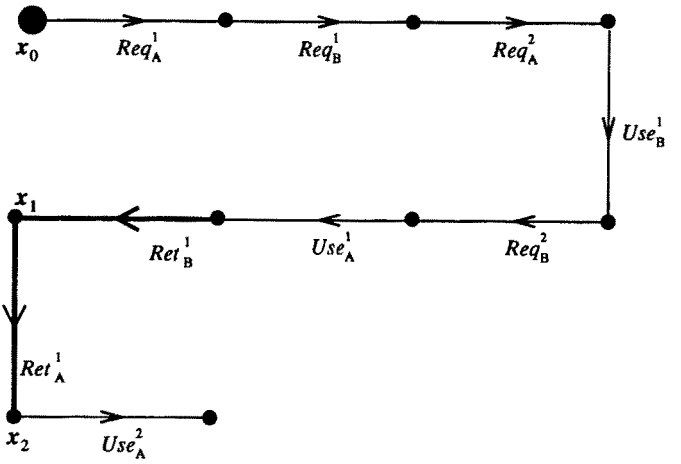

Figure 11. Prioritized solution.

the system under the **VLP-PO** policy partially listed in Table 2 (along with the corresponding orderings). Note that at x_0 the reverse ordering (i.e., User 2 has priority) would enable User 2 to access the resources and deny access to User 1. Also note that at x_1 the ordering is irrelevant (since User 1 did not yet return all his resources and hence cannot really execute any Use events); i.e., the priorities do not have to change to let User 2 proceed. At x_2 we change the priorities (so that User 2 has priority) for the sake of fairness. Note that $\sup CN(K) = \emptyset$ in this case too. Also the first and only control action of the second policy of (Heymann and Lin 1994) disables all the controllable events, thereby denying access to the resources to both users forever.

Remark. Although the **VLP-PO algorithm** always generates a maximal controllable and observable sublanguage, not all maximals may be desirable. However, since computing individual control actions is computationally inexpensive, simulation is an attractive

Table 2. Policy for prioritized solution.

State	Control Action	Ordering
x_0	$\{\text{Use}_A^1, \text{Use}_B^1\}$	$[\text{Use}_A^1, \text{Use}_B^1, \text{Use}_A^2, \text{Use}_B^2]$
x_1	$\{\text{Use}_A^i, \text{Use}_B^i\}$	$[\text{Use}_A^1, \text{Use}_B^1, \text{Use}_A^2, \text{Use}_B^2]$
x_2	$\{\text{Use}_A^2, \text{Use}_B^2\}$	$[\text{Use}_A^2, \text{Use}_B^2, \text{Use}_A^1, \text{Use}_B^1]$

option for determining (off-line) suitable parameters for a particular problem (good observable/controllable subsets and priorities) which guarantee the generation of "good" maximals.

7. Limited Lookahead Version of VLP-PO

When the system to be controlled has a large number of states the **VLP-PO** algorithm given in Section 4 may not be fast enough to meet the required time constraints for generating a new control action; it may not converge at all if the state space is infinite. In such cases we can use limited lookahead policies. In other words, for any given trace $s \in L(G)$, we are given the N-step lookahead window $L(G)/s|_N$ and $K/s|_N$ with state information (see (Ben Hadj-Alouane et al. 1994c)). We then use basically the same **VLP-PO** algorithm given in Section 4 with the following modification: The legality of pending states on the boundary (Nth level states) depends on the attitude adopted (see (Chung et al. 1992), (Ben Hadj-Alouane et al. 1994c)). In this way **VLP-S** is used within **VLP-PO** to determine the cost of the states in *the part* of the current extended unobservable reach that *falls within the lookahead window*; moreover, for such state x, the residual lookahead window size n, which is the difference between the level of x and N, must be included as a variable in the call to **VLP-S**(x) (this version of **VLP-S** is described in (Ben Hadj-Alouane et al. 1994c)).

Further investigation of limited lookahead policies reveals that only the conservative attitude can be adopted under partial observation. The optimistic attitude cannot be used because under the optimistic attitude, the system may unobservably move out of the current lookahead window (this happens if the unobservable reach goes beyond the boundary), and hence cause an ambiguity in determining the next state set, NS, after the occurrence of the next observable event. Under the conservative attitude, all the states on the boundary (Nth level states) are assumed to be illegal and hence the system can never go beyond the boundary (assuming that it can start properly). While within the boundary, **VLP-PO** guarantees that the system will stay inside K. Hence, the following result follows as an immediate corollary of Theorem 2.

COROLLARY 4 *If* $K^{\uparrow} \neq \emptyset$, *then* $L(G, \gamma^N_{p,cons}) \subseteq K$.

With limited lookahead under complete observation, we find that the language resulting from the conservative policy increases monotonically as the window size N increases

(Chung et al. 1992), (Ben Hadj-Alouane et al. 1994c). This is not the case for limited lookahead under partial observation, as demonstrated by the following example.

EXAMPLE 4 Let $\Sigma = \Sigma_{uo} = \Sigma_c = \{\alpha, \beta\}$, $L(G) = \overline{(\alpha+\beta)(\alpha\alpha\alpha\beta+\beta\alpha)}$ and $K = \overline{(\alpha+\beta)(\alpha\alpha\alpha\beta+\beta)}$. If we use the controllable event ordering $\{\alpha, \beta\}$, then $L(G, \gamma^4_{p,cons}) = \overline{\beta\alpha\alpha\alpha\alpha\beta}$ and $L(G, \gamma^5_{p,cons}) = \overline{\alpha\alpha\alpha\alpha\alpha\beta}$. Clearly, $L(G, \gamma^4_{p,cons}) \not\subseteq L(G, \gamma^5_{p,cons})$.

The above example also shows that the limited lookahead policy may not generate maximal controllable and observable sublanguages of K since $L(G, \gamma^2_{p,cons}) = \emptyset$.

An important issue associated with the use of limited lookahead consists of determining a bound on the depth of the lookahead window which guarantees the convergence of the limited lookahead version of **VLO-PO** (in the case of infinite state systems) and the recovery of the policies generated by the unbounded version of **VLP-PO** given in Section 4. Such a bound can be derived by noticing that (1) all the states in the current extended unobservable reach must fall inside the lookahead window and (2) the residual depth of each state in the current extended unobservable reach must allow for its true cost to be determined. Hence proceeding as in (Ben Hadj-Alouane et al. 1994c) we obtain the bound

$$N_{p,B} \stackrel{\text{def}}{=} \max_{x \in X, s \in L(G)/[x]} |w_o(x,s)| + 1 + \max_{x \in X, s \in L(G)/[x]} |w_c(x,s)|,$$

where

$$w_o(x,s) \stackrel{\text{def}}{=} \begin{cases} \{\delta(x,t)\colon t \leq s\} & \text{if } (\forall t \leq s)(\delta(x,t) \in X_H) \wedge s \in \Sigma^*_{uo}, \\ \emptyset & \text{otherwise.} \end{cases}$$

and

$$w_c(x,s) \stackrel{\text{def}}{=} \begin{cases} \{\delta(x,t)\colon t \leq s\} & \text{if } (\forall t \leq s)(\delta(x,t) \in X_H) \wedge s \in \Sigma^*_{uc}, \\ \emptyset & \text{otherwise.} \end{cases}$$

When $N \geq N_{p,B}$ the above **VLP-PO** algorithm behaves exactly as the unbounded version given in Section 4.1; in the case where the state space is infinite, the unbounded **VLP-PO** algorithm converges if $N_{p,B}$ is finite (this has to be proved for the particular system at hand).

8. On-Line Parallel/Distributed Supervisory Control

In the remainder of this paper, we present a parallel/distributed implementation of the **VLP-PO** algorithm that results in significant computational savings. First, we consider the case of distributing the **VLP-PO** algorithm over two communicating agents. In Section 10, we will consider the case of n agents. In both of these cases, we make structural assumptions on G and K. In Section 11, we will make no structural assumptions on G but only assume that K is the intersection of a set of specifications (see Section 11.1). As we shall see, in all of these cases, computational savings can be achieved over the sequential implementation of **VLP-PO**.

In this section we assume G is generated by the shuffle of three finite state machines $G_i = (\Sigma_i, X_i, \delta_i, x_{0i})$, for $i = 1, 2$, and $G_{12} = (\Sigma_{12}, X_{12}, \delta_{12}, x_{012})$, i.e., $G = G_1||G_{12}||G_2$ with $\Sigma_1 \cap \Sigma_2 = \emptyset$ and $\Sigma_i \cap \Sigma_{12} = \emptyset$, $i = 1, 2$. The sets of observable (unobservable) and controllable (uncontrollable) events are, respectively, Σ_o (Σ_{uo}) and Σ_c (Σ_{uc}); all are subsets of $\Sigma = \Sigma_1 \cup \Sigma_{12} \cup \Sigma_2$. We assume that the legal language K is the parallel composition of two prefix-closed legal sublanguages K^1 and K^2, where $K^i \subseteq L(G_i||G_{12})$, $i = 1, 2$. To formally define it, we consider the following two projections, T^i: $\Sigma^* \longrightarrow (\Sigma^i)^*$, for $i = 1, 2$, where $\Sigma^i = \Sigma_i \cup \Sigma_{12}$, and their inverse operators, respectively denoted by $[T^i]^{-1}$. We have,

$$K = K^1||K^2 \stackrel{\text{def}}{=} [T^1]^{-1}(K^1) \cap [T^2]^{-1}(K^2).$$

Hence,

$$s \in K \quad \Leftrightarrow \quad T^1(s) \in K^1 \wedge T^2(s) \in K^2. \tag{6}$$

Obviously, K is prefix-closed. Also, note in passing that K is decomposable (see, e.g., Rudie and Wonham 1992) with respect to T^1 and T^2; i.e.,

$$K = [T^1]^{-1}(T^1(K)) \cap [T^2]^{-1}(T^2(K)) \cap L(G_1||G_{12}||G_2).$$

Each K^i, $i = 1, 2$, is given in terms of the finite state machine $G^i_H = (\Sigma_i \cup \Sigma_{12}, X^i_H, \delta^i_H, x^i_{0H})$. The projections T_i: $\Sigma^* \longrightarrow (\Sigma_i)^*$, $i = 1, 2$ and T_{12}: $\Sigma^* \longrightarrow (\Sigma_{12})^*$ are also used in the following subsections.

In the two following subsections, we first give the parallel/distributed **DI-VLP-PO** algorithm that synthesizes the control policy of the above problem in an on-line fashion, and then present the main properties of the above algorithm and compare its performance, from a computational viewpoint, to its sequential counterpart **VLP-PO**.

8.1. Statement of the Algorithm

The situation of parallel/distributed control is captured in Figure 12, where P is defined as in Section 2. There are two "agents", both observing the same observable events, running on separate processors (or possibly two workstations connected via a network); they concurrently invoke copies of the procedure listed below. Agent 1 accesses G_1, G_{12}, K^1 and invokes **DI-VLP-PO^1**, obtained from **DI-VLP-PO^i**, given below, by setting $i = 1$ and $j = 2$. Agent 2 accesses G_2, G_{12}, K^2 and invokes **DI-VLP-PO^2**, obtained from the procedure given below by setting $i = 2$ and $j = 1$.

The execution of observable events by the system $G_1||G_{12}||G_2$ is monitored by both agents. Each time a new observable event is executed, both agents simultaneously invoke their respective copies of **DI-VLP-PO^i** as indicated above. The local version of **DI-VLP-PO^i** works essentially in the same way as **VLP-PO**. It calls the procedure **Control-Action^i**, also listed below, to participate in computing the new control action. **Control-Action^i** operates on the product machine $G^i \stackrel{\text{def}}{=} (G_i \times G_{12} \times G^i_h) = (\Sigma^i, X^i, \delta^i, x^i_0)$, where G^i_h is obtained from G^i_H (the machine generating K^i) by adding the illegal state, q^i_{il}, via the

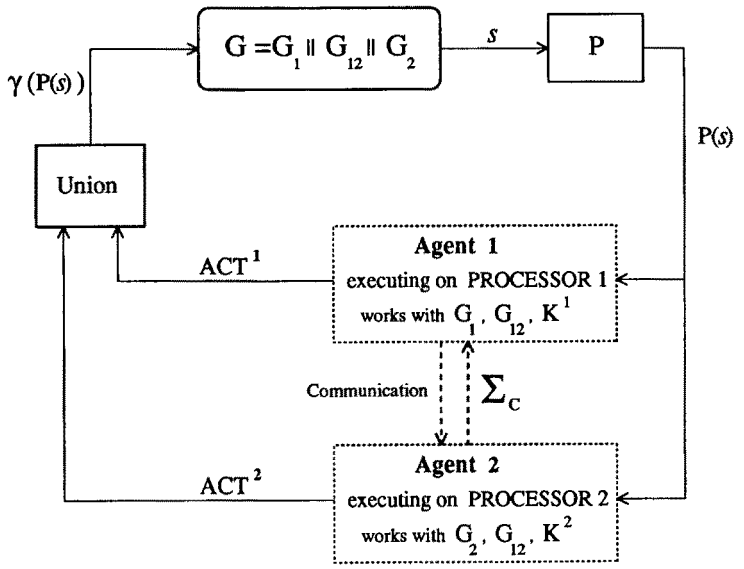

Figure 12. Distributed control under partial observation.

method reviewed in Appendix A. **VLP-S^i**, a version of the algorithm **VLP-S** is used to compute the state costs of G^i with the legal state set, $\{(x, y, z) \in X^i\colon z \neq q_{il}^i\}$.

Both control-action procedures (of agents 1 and 2) work in parallel and in a synchronous fashion. At each new control action computation, both procedures are invoked with the *same* controllable events ordering; i.e. their Elist's start out the same (in terms of the list of events and the pointer). It is proved in the next subsection that following a WHILE loop execution (of both WHILE loops) the Elist's remain the same. Hence, during any WHILE loop execution both procedures examine the same controllable event. Each procedure takes one of four decisions on the event: (1) NEUTRAL, (2) SKIP, (3) DISABLE and (4) ENABLE. Then each procedure sends a message to its counterpart (at the remote agent) indicating the decision and waits (sitting idle) to receive the decision of its counterpart (which might still be busy determining its decision). Steps 2.4–2.6 of each control-action procedure use both decisions to take the same action on the event (that is why the Elist's remain the same throughout the WHILE loop executions). Hence, the message passing scheme together with the operating conditions and Steps 2.4–2.6, ensure that both control-action procedures remain synchronized and examine in parallel the same event. When both control-action procedures terminate, the union of the local ACT^i, $i = 1, 2$, constitutes the new control action, as shown in Figure 12. Note, that due to the workings of the control-action procedures, the ACT^i agree on the Σ_{12} (the shared) events.

The following versions of the operators defined in Section 4 (for **VLP-PO**) are needed (unobservable reach, extended unobservable reach and next state set).

- $UR_E^i(S) \stackrel{\text{def}}{=} \{x \in X^i\colon x = \delta^i(y, u)$ for $y \in S$ and $u \in T^i((\Sigma_{uo} \cap E)^*)\}$.

- $UR+^{i}_{E}(S) \stackrel{\text{def}}{=} UR_{E}(S) \cup \{x \in X^i\colon x = \delta^i(y, \beta)$, for $y \in UR^{i}_{E}(S)$ and $\beta \in T^i((\Sigma_o \cap E))\}$.

- $N^{i}_{\sigma}(S) \stackrel{\text{def}}{=} \{x \in X^i\colon x = \delta^i(y, T^i(\sigma))$ for $y \in S\}$.

The procedures **DI-VLP-PO**^{i} and **Control-Action**^{i} are as follows.

PROCEDURE DI-VLP-PO$^i(\beta_o \in \Sigma_o \cup \{\epsilon\})$;

1. $NS^i := N^{i}_{\beta_o}(PS)$;
2. **Control-Action**^{i}(NS, Event-Ordering);
3. $PS^i := UR^{i}_{ACT}(NS)$;
4. END;

PROCEDURE Control-Action$^i(S \subseteq X^i$, **EList : Ordered-**$\Sigma_c)$;

1. $ACT^i := \Sigma_{uc} \cap (\Sigma_i \cup \Sigma_{12})$; Pt := 1;
2. WHILE [EList $\neq \emptyset$] DO
2.1 IF [Pt > | EList |] THEN
$ACT^i := ACT^i \cup (EList \cap (\Sigma_i \cup \Sigma_{12}))$; RETURN;
2.2 IF [EList.Pt $\in (\Sigma_i \cup \Sigma_{12})$] THEN
{
IF [$(\forall x \in UR^{i}_{ACT^i}(S))\delta^i(x$, EList.Pt) is not defined] THEN
Response := SKIP; GOTO 2.3;
FOR [$x \in UR^{i} +_{ACT^i \cup \{EList.Pt\}} (S)$] DO
{
VLP-$S^i(x)$;
IF [$V^i(x) = \infty$] THEN
Response := DISABLE; GOTO 2.3;
}
Response := ENABLE;
}
ELSE Response := NEUTRAL;
2.3 Send-to-Agent^j(Response); Message := Receive-from-Agent^j();
2.4 IF [Response=DISABLE $\vee$ Message=DISABLE] THEN
EList := EList $-\{$ EList.Pt $\}$; GOTO 2;
2.5 IF [Response=ENABLE $\vee$ Message=ENABLE] THEN
$ACT^i := ACT^i \cup (\{$ EList.Pt $\} \cap (\Sigma_i \cup \Sigma_{12}))$;
EList := EList $-\{$ EList.Pt $\}$; Pt :=1; GOTO 2;
2.6 Pt := Pt+1; GOTO 2;
3. END;

The following conditions are needed for proper operation of **DI-VLP-PO**^{i}.

- $PS^i := \{x^i_0\}$ prior to the first call.

- **Event-Ordering** := a list reflecting the desired ordering for the events of Σ_c. Both calls to **DI-VLP-PO**^{i}, $i = 1, 2$, to compute any control action, *must* use the same ordering.

- **DI-VLP-PO**$^i(\epsilon)$, $i = 1, 2$ are used to determine the first control action.

8.2. *Main Results about the Algorithm*

The first main result of this section (Corollary 5) states that **DI-VLP-PO** computes policies which generate *maximal* controllable and observable sublanguages of the legal language K. Our strategy in deriving this result consists of first showing that, under similar operating conditions, **DI-VLP-PO** computes exactly the same policies as its sequential counterpart **VLP-PO** (Theorem 6). For the purpose of this comparison we assume that **VLP-PO** is operating on $G \stackrel{\text{def}}{=} G_1 \times G_{12} \times G_2 \times G_h^1 \times G_h^2 = (\Sigma, X, \delta, x_0)$ (note that this assumption can be made w.l.o.g. since the policies generated by **VLP-PO** are independent of the actual machine it is given, as long as the machine correctly generates the system and legal language). The second main result (Theorem 7) gives the computational and communication complexities required by **DI-VLP-PO**.

The following is a list of properties of the operators (within the context of the special structure of our problem) used by the two algorithms. These properties capture the structure of the problem and are at the core of the proof of Theorem 6.

Property 1:. $\mathrm{UR}_{\mathrm{E}}(\mathrm{S})|^i = \mathrm{UR}^i_{\mathrm{E}\cap\Sigma^i}(\mathrm{S}|^i)$, $i = 1, 2$.

The operation $|^i$, $i = 1, 2$ denotes the familiar projection from the state space of G onto the state space of G^i (i.e., it gives the G^i components of a state in G). We argue the case $i = 1$ (the case $i = 2$ is similarly argued):

$$\begin{aligned}
\mathrm{UR}_{\mathrm{E}}(\mathrm{S})|^1 = \{ & (x_1, x_{12}, x_2, x_h^1, x_h^2) \in X: \\
& (x_1, x_{12}, x_2, x_h^1, x_h^2) = \delta_1 \times \delta_{12} \times \delta_2 \times \delta_h^1 \times \delta_h^2((y_1, y_{12}, y_2, y_h^1, y_h^2), t) \\
& \text{for some } (y_1, y_{12}, y_2, y_h^1, y_h^2) \in S \text{ and } t \in (\Sigma_{uo} \cap \mathrm{E})^*\}|^1 \\
= \{ & (x_1, x_{12}, x_2, x_h^1, x_h^2) \in X: \\
& (x_1, x_{12}, x_2, x_h^1, x_h^2) = (\delta_1(y_1, T_1(t)), \delta_{12}(y_{12}, T_{12}(t)), \delta_2(y_2, T_2(t)), \\
& \delta_h^1(y_h^1, T^1(t)), \delta_h^2(y_h^2, T^2(t))) \\
& \text{for some } (y_1, y_{12}, y_2, y_h^1, y_h^2) \in \mathrm{S} \text{ and } t \in (\Sigma_{uo} \cap \mathrm{E})^*\}|^1
\end{aligned}$$

[By definition of machine cross-products.]

$$\begin{aligned}
(\subseteq\supseteq) \quad = \{ & (x_1, x_{12}, x_h^1) \in X^1\colon (x_1, x_{12}, x_h^1) = (\delta_1(y_1, T_1(v)), \delta_{12}(y_{12}, T_{12}(v)), \\
& \delta_h^1(y_h^1, T^1(v))) \text{ for some } (y_1, y_{12}, y_h^1) \in \mathrm{S}|^1 \text{ and } v \in (\Sigma_{uo} \cap \mathrm{E} \cap \Sigma^1)^*\}
\end{aligned}$$

[The left inclusion is automatic (take $v = T^1(t)$). For the right inclusion, observe that

$L(h^2) = (\Sigma^2)^*$ and, Σ_2, the alphabet of G_2, is disjoint from Σ^1, the alphabet of G^1. Hence, anything that is executed by G^1 is executed by G, since it cannot be blocked by G_2 or h^2.]

$\stackrel{\text{def}}{=} \mathrm{UR}^1_{E\cap\Sigma^1}(S|^1)$. ∎

Properties 2–4, listed below, can be proved using arguments similar to the above.

Property 2:. $\mathrm{UR}{+_E}(S)|^i = \mathrm{UR}{+^i_{E\cap\Sigma^i}}(S|^i)$, $i = 1, 2$.

Property 3:. $\mathrm{N}_\beta(S)|^i = \mathrm{N}^i_\beta(S|^i)$, $i = 1, 2$.

The following property states that if new behavior can be generated via the event $\sigma \in \Sigma^i$ from $\mathrm{UR}^i_{E\cap\Sigma^i}(S|^i)$ in G^i, $i = 1, 2$, then new behavior can be generated via σ from $\mathrm{UR}_E(S)$ in G.

Property 4:. $[(\exists x^i \in \mathrm{UR}^i_{E\cap\Sigma^i}(S|^i))\delta^i(x^i, \sigma)$ is defined]
$\Rightarrow$
$[\ (\exists x \in \mathrm{UR}_E(S))\delta(x, \sigma)$ is defined $]$, $i = 1, 2$.

We argue the case $i = 1$ (the case $i = 2$ can be similarly argued): Consider $y \in S$ and $t \in (\Sigma^1 \cap \Sigma_{uo})^*$ such that $\delta^1(y|^1, t) = x^1$. Since $L(h^2) = (\Sigma^2)^*$ and $\Sigma_2 \cap \Sigma^1 = \emptyset$, t is not blocked by h^2 or G_2 and can be executed from y in G. Moreover, $\delta(y, t) \stackrel{\text{def}}{=} x \in \mathrm{UR}_E(S)$. For the same reasons that t can be executed from y in G, σ can be executed from x in G; hence, $\delta(x, \sigma)$ is defined. ∎

The following property states that if new behavior can be generated via $\sigma \in \Sigma$ from $\mathrm{UR}_E(S)$ in G, then either new behavior can be generated via σ from $\mathrm{UR}^1_{E\cap\Sigma^1}(S|^1)$ in G^1 *or* new behavior can be generated via σ from $\mathrm{UR}^2_{E\cap\Sigma^2}(S|^2)$ in G^2. Its proof is similar to the above.

Property 5:. $[(\exists x \in \mathrm{UR}_E(S))\ \delta(x, \sigma)$ is defined]
$\Rightarrow$
$[(\exists x^1 \in \mathrm{UR}^1_{E\cap\Sigma^1}(S|^1))\ \delta^1(x^1, \sigma)$ is defined $\vee\ (\exists x^2 \in \mathrm{UR}^2_{E\cap\Sigma^2}(S|^2))\ \delta^2(x^2, \sigma)$ is defined.]

THEOREM 6 *Following the same ordering scheme of controllable events,* **DI-VLP-PO** *and* **VLP-PO** *compute the same control policy.*

By an ordering scheme, we mean a given selection of orderings for Σ_c, for each $s \in P(L(G))$. The proof of the above theorem is given in Appendix C.

The following corollary is an immediate consequence of the above theorem together with Theorem 4.

COROLLARY 5 *Let* $K^\uparrow \neq \emptyset$. *Any control policy computed using* **DI-VLP-PO** *generates a maximal controllable and observable sublanguage of* K.

The following theorem addresses the computational performance of the algorithm.

THEOREM 7 (1) *Under the assumption that a communication and a computation step execute in times that are of the same order, the worst-case execution time of* **DI-VLP-PO** *is in the order of*

$$O(max(|\Sigma_c|^2|X^1|, |\Sigma_c|^2|X^2|, |\Sigma^1||X^1|, |\Sigma^2||X^2|)).$$

(2) *The maximum number of messages exchanged between the two agents of* **DI-VLP-PO** *is* $2|\Sigma_c|^2$.

The term "execution time" above refers to the total time required to compute a control action (including exchange of messages, actual computation and time an agent spends waiting to receive a message from another agent).

Proof: (1) Locally, each agent executes like the sequential **VLP-PO** (operating on G^i) with one exception: At the end of each WHILE loop iteration a communication step is executed. In this step a message is sent by the "local" agent and then the response of the "remote" agent is awaited (if the remote agent is still performing its WHILE loop computations). Hence, in the worst case, the total work performed by each agent in each WHILE loop iteration equals the sum of his WHILE loop work and the remote agent's WHILE loop work (i.e., the situation where the local agent performs his WHILE loop and sends his message while the remote agent is idle and then the remote agent begins his WHILE loop work.) From the above observations and the analysis of the computational complexity of **VLP-PO** in the proof of Theorem 3, it follows that in the worst case an agent terminates within $O(max(|\Sigma_c|^2|X^1|, |\Sigma_c|^2|X^2|, |\Sigma^1||X^1|, |\Sigma^2||X^2|))$. The worst-case execution time of **DI-VLP-PO** is the maximum of the-worst case execution times of both agents, which is also $O(max(|\Sigma_c|^2|X^1|, |\Sigma_c|^2|X^2|, |\Sigma^1||X^1|, |\Sigma^2||X^2|))$.

(2) As indicated above, a communication step is executed at the end of each WHILE loop (a communication step involves two messages across the channel). From the analysis of the computational complexity of **VLP-PO** in the proof of Theorem 3, it follows that the WHILE loop of an agent executes at most $|\Sigma_c|^2$ times. Hence, there are at most $|\Sigma_c|^2$ communication steps; or the maximum number of messages exchanged between the two agents is is $2|\Sigma_c|^2$. ■

To effectively compare the results of the above theorem with the computational complexity of **VLP-PO** which is $O(max(|\Sigma_c|^2|X|, |\Sigma_c||X|))$, we note the following explicit expressions for the sizes of the state spaces: (1) $|X| \leq |X_1||X_{12}||X_2||X_h^1||X_h^2|$ and (2) $|X^i| \leq |X_i||X_{12}||X_h^i|$, $i = 1, 2$ ($\leq$ is used since only the accessible part of a cross-product machines is relevant; for worst-case analysis purposes, however, $\leq$ becomes $=$). Obviously, an order of magnitude of savings is achieved.

9. A Decoupled Resource Allocation Problem

In this section we illustrate the use of **DI-VLP-PO** by applying it to a decoupled version of the resource allocation problem presented in Section 6. The version of the problem

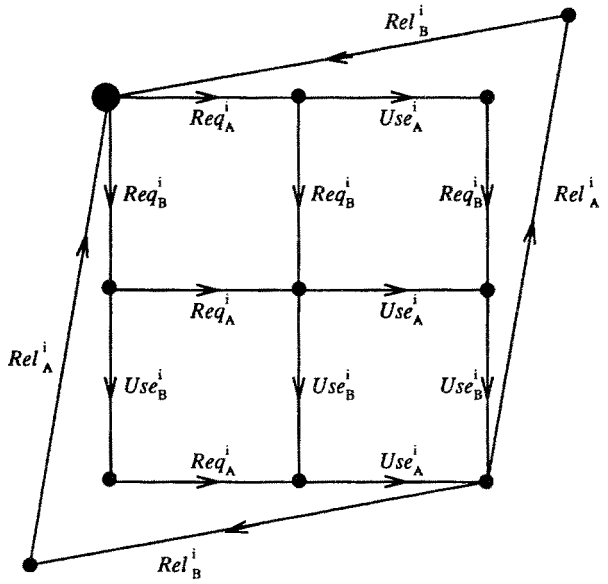

Figure 13. Model of user i.

considered here is decoupled in the sense that the users do not communicate directly with the resource pool or with each other; hence mutual exclusion, among other issues, must be enforced by the supervisor.

The models of user i, denoted by G_i ($i \in \{1, 2\}$), and the resource pool, denoted by G_{12}, are shown, respectively, in Figures 13 and 14. The events Req_A^i, All_A^i, Use_A^i, Rel_A^i and Ret_A^i capture, respectively, a request for resource A by user i (issued to the supervisor), the allocation of resource A to user i (by the resource pool), the actual access of user i to resource A, the releasing of resource A by user i and the actual return of resource A to the resource pool. Note, for example, that the two events All_A^i and Use_A^i above accomplish part of the function of the single event Use_A^i in the version of the problem presented in Section 6 (where User i communicates directly with the resource pool; this communication is captured by the synchronization between the corresponding models). The complete system model G consists of the shuffle of the three models G_1, G_2 and G_{12} (i.e., $G = G_1||G_2||G_{12}$).

With the above model, mutual exclusion is achieved by ensuring that (1) a user only accesses a resource that is already allocated to him and (2) the resource pool only "grabs" resources already released by the users. The above is accomplished by a supervisor with the models partly shown in Figure 15 as part of its legal language. The left model on Figure 15 is denoted AU_A^i and specifies that User i can access resource A only if it is already allocated to him. The right model of Figure 15 is denoted RR_A^i and specifies that resource A must first be released by User i (obviously, the user must be currently accessing it) before it can be returned to the resource pool. There are comparable models for resource B which are not shown in Figure 15.

Note (from the G_i models) that each user must first acquire *both* resources (in any order),

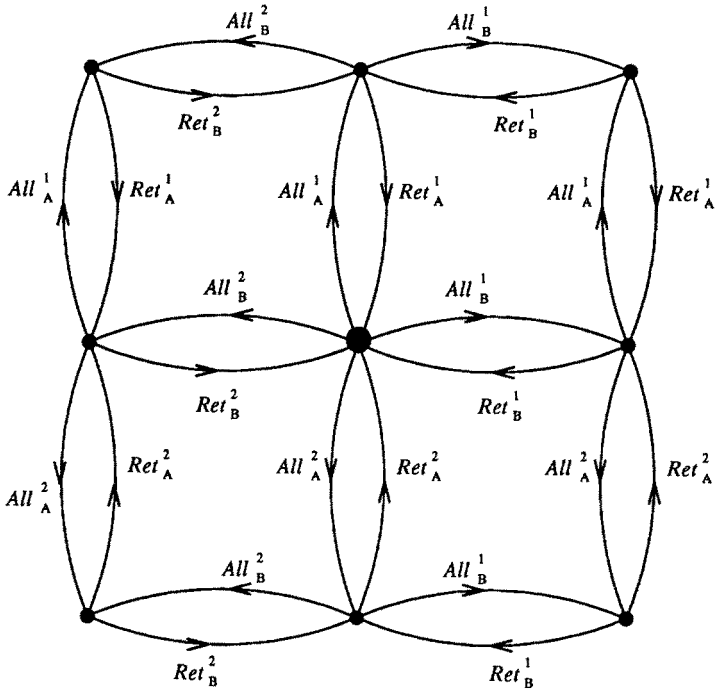

Figure 14. Model of resource pool.

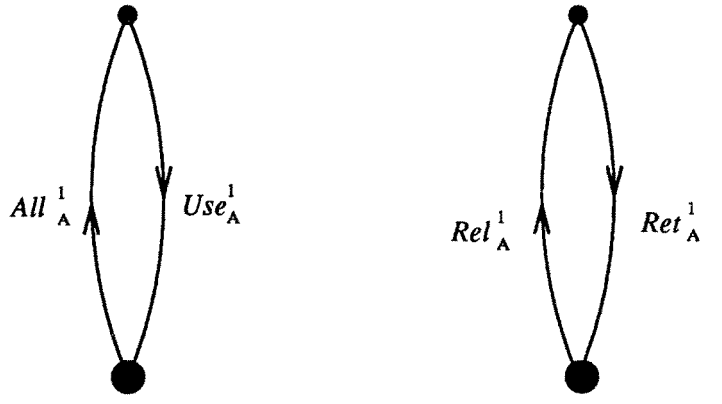

Figure 15. Mutual exclusion specification for user 1, resource *A*.

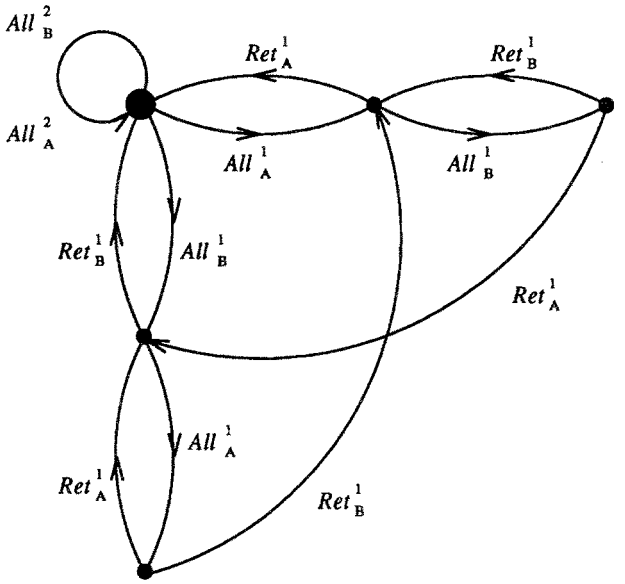

Figure 16. Non-blocking specification for user 1.

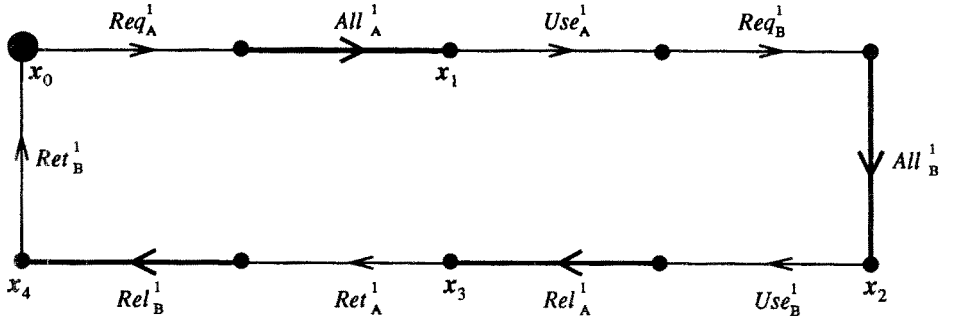

Figure 17. Distributed solution trajectory.

presumably to perform some unmodeled task, before returning them to the pool. Moreover, if a given user is allowed to acquire a particular resource and, at the same time, the other user is allowed to acquire the other resource a "deadlock" situation results. To avoid this, in a manner similar to that of Section 6, B_1 displayed in Figure 16 is used to specify the legal behavior for user 1. B_2 for user 2 can be obtained from B_1 by replacing each event with the superscript 1 in its counterpart with a superscript 2, and vice versa.

Hence, the complete model of the specification for user i is the parallel composition $H^i = AU_A^i||RR_A^i||AU_B^i||RR_B^i||B_i$.

We consider the task consisting of the distributed supervision of the system G for the purpose of achieving a subset of the specified behavior K (such that $K = L(H) = L(H^1||H^2)$) under $\Sigma_c = \{\text{All}_A^i, \text{All}_B^i, \text{Use}_A^i, \text{Use}_B^i, \text{Ret}_A^i, \text{Ret}_B^i\colon\ i = 1, 2\}$ and $\Sigma_o =$

Table 3. Policy for distributed solution.

State	Disabled by	
	Agent 1	Agent 2
x_0	$\text{Use}_A^1, \text{Use}_B^1$	$\text{Use}_A^2, \text{Use}_B^2$
x_1	$\text{Use}_B^1, \text{All}_B^2, \text{Ret}_A^1$	$\text{Use}_B^1, \text{Use}_A^2, \text{Use}_B^2$
x_2	$\text{Ret}_A^1, \text{Ret}_B^1$	$\text{Use}_A^2, \text{Use}_B^2$
x_3	$\text{Use}_A^1, \text{Ret}_B^1$	$\text{Use}_A^2, \text{Use}_B^2$
x_4	$\text{Use}_A^1, \text{Use}_B^1$	$\text{Use}_A^2, \text{Use}_B^2$

$\{\text{All}_A^i, \text{All}_B^i, \text{Rel}_A^i, \text{Rel}_B^i\colon\ i = 1, 2\}$. Hence **DI-VLP-PO** is used with the setup previously discussed in Section 8. The path shown in Figure 17 can be executed by the system under the resulting **DI-VLP-PO** policy partially listed in Table 3 (the ordering of events turns out to be irrelevant; hence, **DI-VLP-PO** computes a single policy).

We cite the following data to compare the execution time of the distributed solution over the sequential one (where the sequential **VLP-PO** is used to solve the above problem). The size of the state space of $G = G_1||G_{12}||G_2$ is 1089 whereas the size of the state space of $G^i = G_i||G_{12}$ is 99 (a ratio of 11). The size of state space of the machine used by Agent i, $G^i||h^i$, is 109 whereas the size of the machine that would be used by the sequential **VLP-PO**, $G||h$, is 1098 (a ratio of 10). Hence, we expect a factor of 10 speedup in going from the sequential to the distributed implementation. Moreover, this speedup factor closely matches the size ratio of the machines used by the agents, versus the global system machine.

10. Generalized Problem Structure

In this section, we extend the distributed approach of Section 8 to the case of n agents, $n \geq 2$. Let G be a system that is the shuffle of m distinct subsystems with mutually disjoint alphabets; i.e., $G = G_1||\cdots||G_m$ (with $G_i = (\Sigma_i, X_i, \delta_i, x_{0i})$). Consider the supervision of G, under partial observation, with a legal language K which is the parallel composition of n sublanguages; i.e., $K = ||_{j\in 1,\ldots,n} K^j$ such that $K^j \subseteq L(||_{i\in I_j} G_i)$, where $I_j \subseteq \{1, \ldots, m\}$, and $K^j = L(G_H^j)$.

The control action, at each step, can be computed by a generalized version of the two agents **DI-VLP-PO** presented in Section 8. This generalized version has one agent for each K^j involved in the above parallel composition constituting K. Agent j operates on $G^j \stackrel{\text{def}}{=} ((||_{i\in I_j} G_i)||G_h^j)$, where G_h^j is obtained from G_H^j, the machine generating K^j, by adding to it an illegal state as described in Appendix A. The functioning of the multi-agent

DI-VLP-PO is basically the same as that of the two agent version: Each agent sees the currently observed event; immediately following this observation, each agent proceeds to compute its part of the next control action. A corresponding version of the following procedure is used by each agent for the above purpose (it is a modified version of the two agent **Control-Action** procedure).

PROCEDURE **Control-Action**$^j(S \subseteq X^j$, **EList : Ordered-**Σ_c);

1 – 2.2 Determine Response in the same way as the two agents version.
2.3 Broadcast(Response);
/* Listen to the broadcasts of all agents; n is the total number of agents */
FOR [$k \in 1 \ldots n$] DO Message^k := Receive-broadcast-message();
2.4 IF [$(\exists k, 1 \leq k \leq n)$ Message^k =DISABLE] THEN
EList := EList −{ EList.Pt }; GOTO 2;
2.5 IF [$(\exists k, 1 \leq k \leq n)$ Message^k =ENABLE] THEN
ACT^j := ACT$^j \cup$({ EList.Pt } $\cap$ ($\bigcup_{i \in I_j} \Sigma_i$));
EList := EList −{ EList.Pt }; Pt :=1; GOTO 2;
2.6 Pt := Pt+1; GOTO 2;
3. END;

In Step 2.4 the current event is disabled if any of the computations performed by the agents determine that it must be disabled (it must have generated an illegal state in the current unobservable reach of some agent). In Step 2.5 (none of the agents determined that the event must be disabled) the current event is enabled if any of the agents enables it (it must have generated new behavior in the unobservable reach of some agent). When Step 2.5 is reached the event has not generated any new behavior in the unobservable reach of any agent and is, therefore, skipped.

Following the termination of every **Control-Action**^{j} procedure, the next control action is determined as in the two agent case:

$$\text{ACT} = \left(\bigcup_{j \in 1,\ldots,n} \text{ACT}^j \right).$$

The proof of correctness of the multi-agent **DI-VLP-PO** closely follows the proof of correctness of the two agents version presented in Section 8 (it involves showing that the multi-agent version generates the exact same policy as the sequential **VLP-PO**). Properties 1–5 are easily extended to the case of n agents (since the underlying problem structure is preserved), and the main argument of the proof of Theorem 6 is preserved. Therefore, for purposes of brevity we will not present the details of this proof.

The maximum **number of messages** exchanged between the agents at each execution of the multi-agent **DI-VLP-PO** (i.e., following an observable event) is determined as follows. Following the consideration of the event at the top of Elist, 1 message is broadcast across the channel by each agent. Since the maximum number of times the top of Elist is examined is $|\Sigma_c|^2$ (see proof of Theorem 7), the maximum number of messages sent across the channel during a single control action computation is $n|\Sigma_c|^2$.

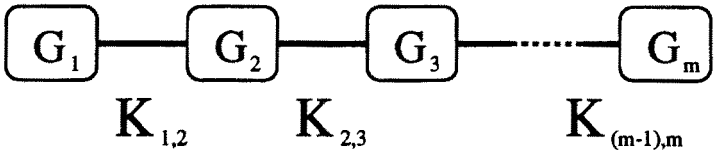

Figure 18. Chain structure.

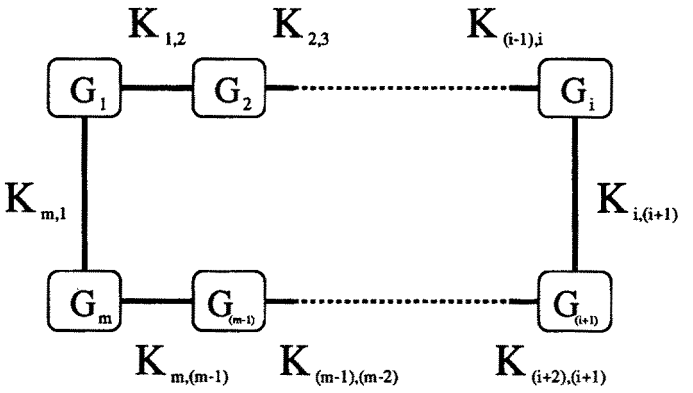

Figure 19. Mesh structure.

We note however, that the number of exchanged messages can be reduced by keeping a table (available to each agent) indicating the number of agents "responsible" for each event (an agent is responsible for the events in its alphabet). A given agent broadcasts its decision only if it is responsible for the current event (an agent always responds NEUTRAL to an event that does not belong to its alphabet and its response does not contribute to the final decision regarding the event). Moreover, the above table determines the number of messages to be received by each agent in order to decide on the current event.

Under the assumptions that a communication step is of the same order as a computation step and that n is smaller than (or of the same order as) the largest state space of any agent subsystem, G^j, the **total execution time** of the algorithm (and this includes actual computation, exchanging of messages and idle time) can be determined as

$$O(\max(|\Sigma_c|^2|X^1|, \ldots, |\Sigma_c|^2|X^n|, |\Sigma^1||X^1|, \ldots, |\Sigma^n||X^n|)),$$

where X^j is the state space of G^j (see proof of Theorem 7).

Figures 18, 19 and 20 show three interesting special structures, respectively, a chain, a mesh and a star, where the number of messages sent across the channel is significantly reduced by the event-agent method outlined above. In the case of the chain and the mesh structures the event currently at the top of Elist belongs to the alphabet of exactly 2 agents. Hence the maximum number of messages exchanged during any control action computation is $2|\Sigma_c|^2$. In the case of the star structured system the event at the top of Elist either belongs to the alphabet of 1 agent (the case where it is in the alphabet of a given G_i) or it belongs to the alphabet of all agents (the case where it is in the alphabet of G_1, the common interface).

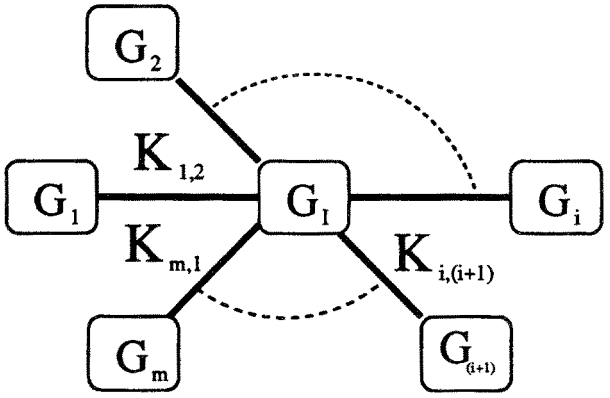

Figure 20. Star structure.

In the first case only one message needs to be sent and in the second case n messages have to be sent.

11. Special Cases

In the following two subsections, we make some observations regarding two special cases of the control problem considered in this paper: (1) the problem has a modular structure and (2) the system is fully observed.

11.1. Modular Control with Partial Observations

Consider the special case where the entire system consists of a single machine G (i.e., not the shuffle of many submachines) and the specification is, as in the general case, $K = ||_{i\in 1,\ldots,n} K^i$, with $K^i \subseteq L(G)$; hence, $K = \bigcap_{i\in 1,\ldots,n} K^i$. This corresponds to the usual situation of *modular control* (see, e.g., (Wonham and Ramadge 1988)), with the important difference that the system G is partially observed. By the results in the previous section, the multi-agent **DI-VLP-PO** algorithm will produce a maximal controllable and observable sublanguage of K.

Suppose now that we were to solve the same problem in an on-line "decentralized/modular-like" manner; i.e., first by having Agent i, $i = 1, \ldots, n$ run the sequential **VLP-PO** algorithm operating on G with a legal sublanguage K^i (without any communication between the agents) and second by taking the intersection of the n control actions so obtained, as in modular control (again, the agents do not communicate their respective control actions to each other; hence each agent "thinks" that the control action he computed is actually the one issued). Then each agent will generate a maximal controllable and observable sublanguage of its given legal sublanguage, and the resulting behavior is the intersection

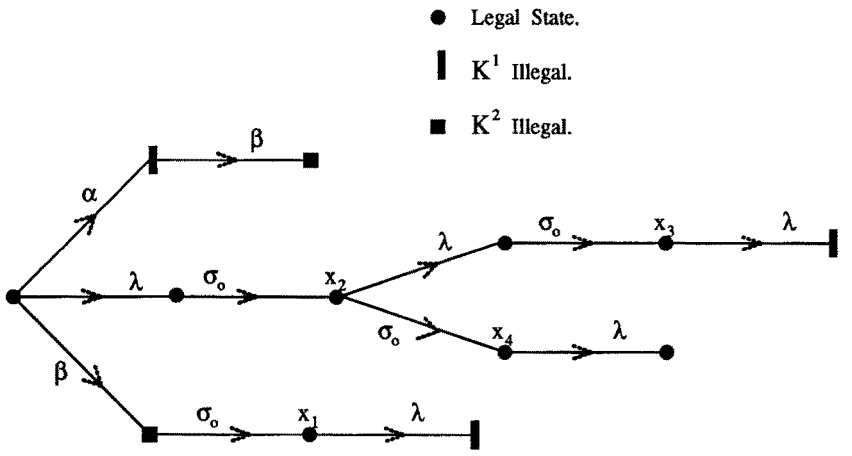

Figure 21. Control under partial observation.

of all these maximals. Clearly, this generated behavior need not be a maximal controllable and observable sublanguage of K.

As it turns out, the behaviors generated by the above two methods (i.e., the multi-agent **DI-VLP-PO** and the modular/decentralized algorithm) can be incomparable even when the same ordering of controllable events is used. This is illustrated by the following example.

EXAMPLE 5 Consider the system G depicted in Figure 21; K^1 and K^2 are also shown in this figure. All the events are unobservable with the exception of σ_o. All the events are controllable. The ordering of the controllable events used to synthesize all control actions is as follows: α, β, λ, and σ_o.

At the initial step, in the case of on-line decentralized/modular control, Agent 1 will enable β, λ, and σ_o while Agent 2 will enable α, λ, and σ_o. Thus overall only λ and σ_o are enabled. It is easily verified that **DI-VLP-PO** will have the same initial control action. After event σ_o is observed, Agent 1 in the decentralized/modular scheme will have $\{x_1, x_2\}$ as its current state estimate while Agent 2 in the decentralized/modular scheme and both agents in the case of the **DI-VLP-PO** algorithm will know that the state of the system is x_2. Thus at the second step, in the case of decentralized/modular control, Agent 1 will enable σ_o, α, and β while Agent 2 will enable λ, σ_o, α, and β; overall α, β, and σ_o are enabled. **DI-VLP-PO** on the other hand will enable λ, σ_o, α, and β. Thus after the second σ_o is executed, the state estimates will be as follows: $\{x_4\}$ for Agent 1 in decentralized/modular scheme; $\{x_3, x_4\}$ for Agent 2 in decentralized/modular scheme; $\{x_3, x_4\}$ for both agents in **DI-VLP-PO**. Finally, at the third step, in the case of decentralized/modular control, Agents 1 and 2 will both enable λ, α, β, and σ_o while **DI-VLP-PO** will enable α, β, and σ_o.

Thus the trace $\lambda\sigma_o\lambda\sigma_o$ is in the behavior generated by the multi-agent **DI-VLP-PO** but not in the behavior generated by the decentralized/modular algorithm. Contrarily, the trace $\lambda\sigma_o\sigma_o\lambda$ is in the behavior generated by the decentralized/modular algorithm but not in the behavior generated by the multi-agent **DI-VLP-PO**. This shows that the two generated languages generated are incomparable.

11.2. Supervision with Complete Observation

Our multi-agent **DI-VLP-PO** algorithm is designed for supervision under partial event observation. However, it is interesting to consider the workings of **DI-VLP-PO** when all events are observed. In this case, some steps in **DI-VLP-PO** and **Control-Action**^{i} can be modified to reduce computation. First, NS^i and PS^i are always singletons. Also, $N^i_\sigma(.)$ and $UR^i_{ACT}(.)$ become identity mappings; and $UR{+}^i_{ACT}(.)$ reduces to the set of next states (via an event in ACT). Furthermore, if an event does not generate new behavior from a given NS^i (i.e., state) while being examined at a particular iteration, then, obviously, it never will generate behavior during any future examination. Hence there is no point to skip the event: it can be enabled right away. In other words, Step 2.2 of **Control-Action**^{i} can be replaced by

```
2.2  IF [ EList.Pt ∈ (⋃_{i∈I_i} Σ_i) ] THEN
     {
      FOR [ x ∈ UR^i +_{ACT^i ∪ {EList.Pt}} (S) ] DO
      {
        VLP-S^i(x);
        IF [ V^i(x) = ∞ ] THEN
           Response := DISABLE; GOTO 2.3;
      }
      Response := ENABLE;
     }
     ELSE Response := NEUTRAL;
```

Since there is no SKIP step, the WHILE loop will be executed exactly $|\Sigma_c|$ times. Hence the worst-case execution complexity of the modified **DI-VLP-PO** is

$$O(max(|\Sigma_c||X^1|, \ldots, |\Sigma_c||X^n|, |\Sigma^1||X^1|, \ldots, |\Sigma^n||X^n|)).$$

In comparison, the worst-case computational complexity of a sequential procedure using **VLP-S** is $O(|\Sigma||X|)$. Therefore, parallel/distributed supervision is still advantageous over centralized supervision. The maximum number of messages exchanged by the modified **DI-VLP-PO** is $(n+1)|\Sigma_c|$. Another important difference is that, since all events are observable, the maximal controllable and observable sublanguage is unique, which is the supremal controllable sublanguage. Therefore, the ordering of controllable events is of no importance: any ordering will result in the supremal controllable sublanguage.

Observe that communication between the agents *during* the computation of the control actions is not really required due to complete event observation. Optimal behavior could be generated by deriving the global control action from the ACT's of the agents computed without communication (e.g. by assuming that the remote agents always sends "NEUTRAL"): $ACT = \bigcup_{i\in 1,\ldots,n}(ACT^i \cap \Sigma^i)$.

Finally note that when all the events are observed *and* the entire system consists of a single machine G (i.e., not a shuffle) we are in the usual situation of modular control, as considered in (Wonham and Ramadge 1988). In this case, the modified version of the multi-agent **DI-VLP-PO** discussed above, becomes an on-line version with communication, of the off-line

solution in (Wonham and Ramadge 1988): The intersection of the control actions that has to be performed after each event execution (in a standard on-line implementation of the solution in (Wonham and Ramadge 1988)) can in some sense be considered a communication step analogous to the communication via exchange of messages in **DI-VLP-PO**.

12. Conclusion

Our approach in this paper has been to address the control of partially-observed DES from an on-line viewpoint and using an event ordering (or priority) scheme for the on-line calculation. The on-line technique, an extension of (Ben Hadj-Alouane et al. 1994c) (for fully observed DES), allows for the computation of a control action in linear complexity in the size of the state space, an important result for the practical usefulness of this on-line technique. The ordering scheme, which improves on prior work (Heymann and Lin 1994), guarantees that maximal controllable and observable sublanguages are generated (unless we are in the limited lookahead situation discussed in Section 7), and makes it possible to generate solutions that include the supremal controllable and normal sublanguage. We believe that this is an interesting and useful contribution.

We also proposed a new framework for distributed control with communication. By exploiting the structure of the system and of the legal language, we derived a general parallel/distributed algorithm for on-line control under partial observations that possesses interesting properties. Such properties include the computational savings made possible by the parallelism in the algorithm and the guaranteed maximality of the resulting controlled behavior.

Recently, *nondeterministic* supervisors have been considered in (Inan 1994) for the control of partially-observed deterministic systems; the supervisors in that reference nondeterministically "generate" a behavior that is the *union* of maximal controllable and observable sublanguages (note that these nondeterministic supervisors allow for all possible control policies to be potentially implemented). The chosen focus of our paper was deterministic supervision and the associated computational issues; however, we can easily extend our approach to a nondeterministic setting as in (Inan 1994), if we adopt a strategy that *randomly* selects the event ordering after each occurrence of observable events.

A Preprocessing of the System and Specification Models

Our starting point is a system model G_o and a specification model S such that $L(S) = K_s$; the legal language K is defined in terms of specification K_s as $K = K_s \cap L(G_o)$ (note that as a matter of modeling convenience, the specification may or may not be a sublanguage of the system language). Our goal is to produce a machine G which is a refinement of G_o and such that a model of the legal language K is a submachine of G (this labels each state of G as either legal or illegal).

Let $G_o = (\Sigma, X, \delta, x_0, X_m)$ and $S = (\Sigma, X_S, \delta_S, x_{0S}, X_{mS})$, with $L(S) = K_s$ (we assume $X_S = X_{mS}$).

First, S is modified by adding to it a new state, x_{il} (for *illegal* state), by completing its transition function and by marking every one of its states. This results in $h = (\Sigma, X_h, \delta_h, x_{0h}, X_{mh})$, where $x_{0h} = x_{0S}$, $X_h = X_S \cup \{x_{il}\}$, $X_{mh} = X_h$ and δ_h is the *total* transition function defined by

$$\delta_h(x, \sigma) = \begin{cases} \delta_S(x, \sigma) & \text{if } \delta_S(x, \sigma) \text{ is defined,} \\ x_{il} & \text{otherwise.} \end{cases}$$

Observe that $L(h) = \Sigma^*$ and $K_s = L(S) = \{t \in L(h)\colon \delta_h(x_{0h}, t) \neq x_{il}\}$.

The desired machine G is the cross-product $G_o \times h$. Observe that $L(G_o) = L(G)$. It is not difficult to see that K is marked by a submachine of G: $K = \{t \in L(G)\colon \delta \times \delta_h((x_0, x_{0h}), t) = (x, y)$, such that $y \neq x_{il}$ and $x \in X_m\}$. Moreover, the "illegal" states of G are readily available: They are all the states that have x_{il} as their second component.

B Proof of Theorem 2

The fact that $L(G, \gamma_p)$ is controllable and observable follows immediately from γ_p being a partial observation policy. This is guaranteed by **VLP-PO** as follows: (1) Following each execution of **VLP-PO**, the value of ACT becomes the new control action; since **VLP-PO** is only executed following an observable event, γ_p is truly a mapping from $P(L(G))$ into 2^{Σ} (i.e., $L(G, \gamma_p)$ is observable); (2) Step 1 of **Control-Action** and the fact that events are never removed from ACT during the the execution of **Control-Action** ensure that $\Sigma_{uc} \subseteq \gamma_p(s)$, for the current observed trajectory s (i.e., $L(G, \gamma_p)$ is controllable).

It remains for us to prove that $L(G, \gamma_p) \subseteq K$. Our proof is based on the following two equations,

$$\mathrm{UR}_{\mathrm{ACT}}(\mathrm{NS}_s) = \{x \in X\colon \delta(x_0, t) = x, \text{ for } t \in L(G, \gamma_p) \cap P^{-1}(s)\} \tag{7}$$

$$(\forall x \in \mathrm{UR} +_{\mathrm{ACT}} (\mathrm{NS}_s))\ V(x) = 0 \tag{8}$$

which we prove to hold following each call to **VLP-PO** issued upon the observation of a given $s \in P(L(G, \gamma_p))$. NS_s denotes the NS computed at Step 1 of the above mentioned **VLP-PO** call. This proof is by induction on the length of s.

Induction Basis: The case where $s = \epsilon$. From the initial conditions of **VLP-PO**, we have $\mathrm{NS}_\epsilon = \{x_0\}$. The ACT computed by **VLP-PO**(ϵ) is by definition $\gamma_p(\epsilon)$.

Equation (7) holds since,

$$\begin{aligned} \mathrm{UR}_{\gamma_p(\epsilon)}(\{x_0\}) &\stackrel{\text{def}}{=} \{x \in X\colon \delta(x_0, t) = x \text{ for } t \in (\gamma_p(\epsilon) \cap \Sigma_{uo})^*\} \\ &= \{x \in X\colon \delta(x_0, t) = x \text{ for } t \in L(G, \gamma_p) \cap P^{-1}(\epsilon)\}. \end{aligned}$$

Equation (8) holds since from the hypothesis of the theorem, we have

$$
\begin{aligned}
K^{\uparrow} \neq \emptyset \;&\Rightarrow\; \epsilon \in K^{\uparrow} && [K^{\uparrow} \text{ is closed}] \\
&\Rightarrow\; \Sigma_{uc}^{*} \cap L(G) \subseteq K && [\text{by controllability of } K^{\uparrow}] \\
&\Rightarrow\; (\forall t \in \Sigma_{uc}^{*} \cap L(G))(\Sigma_{uc}^{*} \cap L(G)/[\delta(x_0, t)]) \\
&\quad \subseteq\; K/[\delta(x_0, t)] \\
&\Rightarrow\; (\forall t \in \Sigma_{uc}^{*} \cap L(G))K^{\uparrow}/[\delta(x_0, t)] \neq \emptyset && [\epsilon \in (\Sigma_{uc}^{*} \cap L(G)/[\delta(x_0, t)])] \\
&\Rightarrow\; (\forall t \in \Sigma_{uc}^{*} \cap L(G))V(\delta(x_0, t)) = 0 && [\text{by Lemma 1}] \\
&\Rightarrow\; (\forall x \in \mathrm{UR} +_{\Sigma_{uc}} (\mathrm{NS}_{\epsilon}))V(x) = 0.
\end{aligned}
$$

The above implies that just before **Control-Action** executes the first WHILE LOOP iteration (Step 2) we have $(\forall x \in \mathrm{UR} +_{\mathrm{ACT}} (\mathrm{NS}_{\epsilon}))V(x) = 0$ (ACT is initialized to Σ_{uc} in Step 1 of **Control-Action**). This fact is obviously preserved throughout the execution of the WHILE LOOP (see Step 2.3). Hence, when **Control-Action** terminates we have $(\forall x \in \mathrm{UR} +_{\mathrm{ACT}} (\mathrm{NS}_{\epsilon}))V(x) = 0$.

Induction Step: We assume that equations (7) and (8) hold for some $s \in P(L(G, \gamma_p))$ and prove that they hold for $s\beta \in P(L(G, \gamma_p))$, where $\beta \in \Sigma_o$.

Equation (7) holds for $s\beta$ since,

$$
\begin{aligned}
\mathrm{UR}_{\mathrm{ACT}}(\mathrm{NS}_{s\beta}) &\stackrel{\mathrm{def}}{=} \mathrm{UR}_{\gamma_p(s\beta)}(\mathrm{N}_{\beta}(\mathrm{PS}_s)) \\
&= \mathrm{UR}_{\gamma_p(s\beta)}(\mathrm{N}_{\beta}(\mathrm{UR}_{\gamma_p(s)}(\mathrm{NS}_s))) \\
&= \mathrm{UR}_{\gamma_p(s\beta)}(\mathrm{N}_{\beta}(\{x \in X\colon \delta(x_0, t) = x \text{ for } t \in L(G, \gamma_p) \cap P^{-1}(s)\})) \\
&\quad [\text{by (7) and ind. hyp.}] \\
&= \mathrm{UR}_{\gamma_p(s\beta)}(\{x \in X\colon \delta(x_0, t) = x \text{ for } t \in L(G, \gamma_p) \cap P^{-1}(s)\beta\}) \\
&= \{x \in X\colon \delta(x_0, t) = x \text{ for } t \in L(G, \gamma_p) \cap P^{-1}(s\beta)\}.
\end{aligned}
$$

The proof that Equation (8) holds for $s\beta$ is as follows. Since Since $\mathrm{NS}_{s\beta} \subseteq \mathrm{UR} +_{\gamma_p(s)} (\mathrm{NS}_s)$, the induction hypothesis implies that $(\forall x \in \mathrm{NS}_{s\beta})V(x) = 0$. But for $x \in \mathrm{NS}_{s\beta}$,

$$
\begin{aligned}
V(x) = 0 \;&\Rightarrow\; K^{\uparrow}/[x] \neq \emptyset && \text{[by Lemma 1]}\\
&\Rightarrow\; \epsilon \in K^{\uparrow}/[x] && [K^{\uparrow}/[x] \text{ is closed}]\\
&\Rightarrow\; \Sigma_{uc}^{*} \cap L(G)/[x] \subseteq K/[x] && \text{[by controllability of } K^{\uparrow}/[x]\text{]}\\
&\Rightarrow\; (\forall t \in \Sigma_{uc}^{*} \cap L(G)/[x])\\
&\quad \times (\Sigma_{uc}^{*} \cap L(G)/[\delta(x,t)])\\
&\subseteq\; K/[\delta(x,t)]\\
&\Rightarrow\; (\forall t \in \Sigma_{uc}^{*} \cap L(G)/[x])K^{\uparrow}/[\delta(x,t)] \neq \emptyset && [\epsilon \in (\Sigma_{uc}^{*} \cap L(G)/[\delta(x,t)])]\\
&\Rightarrow\; (\forall t \in \Sigma_{uc}^{*} \cap L(G)/[x])V(\delta(x,t)) = 0 && \text{[by Lemma 1]}\\
&\Rightarrow\; (\forall x \in \mathrm{UR} +_{\Sigma_{uc}} (\mathrm{NS}_{s\beta}))V(x) = 0.
\end{aligned}
$$

The above implies that just before **Control-Action** executes the first WHILE LOOP iteration (Step 2) we have $(\forall x \in \mathrm{UR} +_{\mathrm{ACT}} (\mathrm{NS}_{s\beta}))V(x) = 0$. As mentioned above, this fact is obviously preserved throughout the execution of the WHILE LOOP. Hence, **Control-Action** terminates with $(\forall x \in \mathrm{UR} +_{\mathrm{ACT}} (\mathrm{NS}_{s\beta}))V(x) = 0$.

Now we use equations (7) and (8) to argue that $L(G, \gamma_p) \subseteq K$. Given $t \in L(G, \gamma_p)$ we have using Equation (7),

$$
\begin{aligned}
(\forall v \leq t)\delta(x_0, v) \in \mathrm{UR}_{\gamma(P(v))}(\mathrm{NS}_{P(v)}) \;&\Rightarrow\; (\forall v \leq t)V(\delta(x_0, v)) = 0\\
&\quad \text{[by Equation (8)]}\\
&\Rightarrow\; (\forall v \leq t)v \in K^{\uparrow}\\
&\quad \text{[by Theorem 1 and since } \epsilon \in K^{\uparrow}\text{]}\\
&\Rightarrow\; t \in K.
\end{aligned}
$$

This completes the proof.

C Proof of Theorem 6

Let γ_p be the control policy computed by **VLP-PO** and γ_d be the control policy computed by **DI-VLP-PO**. We prove that $\gamma_p(s) = \gamma_d(s)$ for all $s \in P(L(G, \gamma_p))$. The fact that the computed parts of γ_p and γ_d are identical immediately follows.

Our proof consists of showing that the following sets of equalities always hold after the computation of the control actions at $s \in P(L(G, \gamma_p)$, **by induction** on the length of s:

(i) $\mathrm{NS}|^{i} = \mathrm{NS}^{i}$ for $i = 1, 2$

(ii) $\mathrm{ACT} = \mathrm{ACT}^{1} \cup \mathrm{ACT}^{2}$, i.e. $\gamma_p(s) = \gamma_d(s)$

(iii) $\mathrm{PS}|^{i} = \mathrm{PS}^{i}$ for $i = 1, 2$.

Induction Basis ($s = \epsilon$)**:** Recall the following operating conditions. (1) ACT is obtained via the call **VLP-PO**(ϵ) with PS=$\{(x_{01}, x_{012}, x_{02}, x_{0h^1}, x_{0h^2})\}$. (2) ACT^1 is obtained via the call **DI-VLP-PO**$^1(\epsilon)$ with $\text{PS}^1=\{(x_{01}, x_{012}, x_{0h^1})\}$. (3) ACT^2 is obtained via the call **DI-VLP-PO**$^2(\epsilon)$ with $\text{PS}^1=\{(x_{012}, x_{02}, x_{0h^2})\}$. (4) The above calls are executed using the *same* Σ_c-ordering.

Set of equalities **(i)**: Immediately following Step 1 of each vlp-po procedure (and since all three calls are issued with $\beta_o = \epsilon$) we have, PS=NS and $\text{PS}^i=\text{NS}^i$, for $i = 1, 2$. Also, from the operating conditions, we have $\text{PS}^i=\text{PS}|^i$, for $i = 1, 2$. Hence, it follows that $\text{NS}^i=\text{PS}^i=\text{PS}|^i=\text{NS}|^i$, for $i = 1, 2$.

Equality **(ii)**: We prove, by induction (i.e., we now have two nested inductions) that prior to the nth iteration of the WHILE loops of the three control-action procedures, the three EList's are identical (i.e. they contain the same events ordered in the same way and point to the same event) *and* equality (ii) holds.

Clearly, the above holds prior to iteration 1 of the three WHILE loops: Each EList is set to Σ_c ordered in the same way and points to the first event (Step 1 of each control-action procedure initializes Pt to 1). Also, by Step 1 of the three control-action procedures, $\text{ACT}=\Sigma_{uc}$ and $\text{ACT}^1 \cup \text{ACT}^2 = (\Sigma_{uc} \cap (\Sigma_1 \cup \Sigma_{12})) \cup (\Sigma_{uc} \cap (\Sigma_2 \cup \Sigma_{12})) = \Sigma_{uc}$.

Assume the above holds prior to the nth iteration of the three WHILE loops. Let ELists point to the same event, σ. We show that the control-action procedures of the two agents and **Control-Action** (of **VLP-PO**) take the same action on σ (i.e., ENABLE, DISABLE or SKIP). As a result the above also holds at the end of the nth iteration, or prior to iteration $n + 1$ of the three WHILE loops. There are three distinct cases.

– Case 1, σ is disabled by agents 1 and 2: One of the agents must have executed "Response:=DISABLE" in Step 2.2 of its control-action procedure. Assume it is Agent 1 (the other case is similar). Then $\sigma \in \Sigma^1$ and,

$$(\exists x^1 \in \text{UR} +^1_{\text{ACT}^1 \cup \{\sigma\}} (\text{NS}^1)) V^1(x^1) = \infty.$$

By Property 2, (since $(\text{ACT} \cup \{\sigma\}) \cap \Sigma^1 = \text{ACT}^1 \cup \{\sigma\}$) we have,

$$(\exists x \in \text{UR} +_{\text{ACT} \cup \{\sigma\}} (\text{NS}))\ \ x|^1 = x^1.$$

Let t be the string such that $\delta(x_0, t) = x$. Then, $\delta^1(x_0^1, T^1(t)) = x^1$. By Lemma 1,

$$\begin{aligned} V^1(x^1) = \infty &\Rightarrow (\exists u \in (\Sigma_{uc} \cap \Sigma^1)^*)\ T^1(t)u \in L(G^1) \wedge T^1(t)u \notin K^1 \\ &\Rightarrow (\exists u \in (\Sigma_{uc} \cap \Sigma^1)^*)\ tu \in L(G) \wedge T^1(t)u \notin K^1 \\ &\Rightarrow (\exists u \in \Sigma_{uc}^*)\ tu \in L(G) \wedge tu \notin K \\ &\Rightarrow V(x) = \infty. \end{aligned}$$

Since σ was disabled by Agent 1, σ must generate new behavior from $\text{UR}^1_{\text{ACT}^1}(\text{NS}^1)$ in G^1 (i.e., σ is not skipped by **Control-Action**^{1}). From the induction hypothesis, we have $\text{NS}^1 = \text{NS}|^1$; hence, by Property 4, σ also generates new behavior from $\text{UR}_{\text{ACT}}(\text{NS})$ in G.

Therefore, Step 2.3 of the procedure **Control-Action** is executed at the nth iteration of the WHILE loop and **VLP-PO** takes the same action as the two agents on σ.

– **Case 2, σ is enabled by agents 1 and 2:** One of the two agents has executed "Response:=ENABLE" in Step 2.2 of its control-action procedure and no agent could execute "Response:= DISABLE". Assume it is Agent 1 (the other case is similar). Hence, σ has generated new behavior from $\mathrm{UR}^1_{\mathrm{ACT}^1}(\mathrm{NS}^1)$ in G^1. From the induction hypothesis, we have, $\mathrm{NS}^1 = \mathrm{NS}|^1$; hence, by Property 4, σ generates behavior from $\mathrm{UR}_{\mathrm{ACT}}(\mathrm{NS})$ in G. We show that **Control-Action** takes the same action as the two agents on σ, i.e., σ is not disabled. Assume this is not the case. So, **Control-Action** disables σ, in which case,

$$(\exists x \in \mathrm{UR} +_{\mathrm{ACT}\cup\{\sigma\}} (\mathrm{NS}))V(x) = \infty.$$

Let t be the string such that $\delta(x_0, t) = x$. By Lemma 1,

$$\begin{aligned} V(x) = \infty \;&\Rightarrow\; (\exists u \in \Sigma^*_{uc})\; tu \in L(G) \wedge tu \notin K \\ &\Rightarrow\; (\exists u \in \Sigma^*_{uc})\; T^1(tu) \in L(G^1) \wedge T^2(tu) \\ &\in\; L(G^2) \wedge (T^1(tu) \notin K^1 \vee T^2(tu) \notin K^2) \end{aligned}$$

[The last implication follows from the contrapositive to equation (6), at the beginning of Section 8.]

Consider first the case where $T^1(tu) \notin K^1$. Observe that $\delta^1(x_0|^1, T^1(t)) = x|^1 \in \mathrm{UR}+_{\mathrm{ACT}}(\mathrm{NS})|^1$. By Property 2, $\mathrm{UR}+_{\mathrm{ACT}}(\mathrm{NS})|^1 = \mathrm{UR}^1 +_{\mathrm{ACT}} (\mathrm{NS}|^1)$. Hence, $x|^1 \in \mathrm{UR}^1 +_{\mathrm{ACT}} (\mathrm{NS}^1)$. Since $T^1(t)T^1(u) \in L(G^1) \wedge T^1(t)T^1(u) \notin K^1$, it follows that $V^1(x|^1) = \infty$. Hence, Agent 1 would have to execute "Response:=DISABLE", which contradicts our assumption. In the second case, a similar argument leads to Agent 2 executing "Response:=DISABLE", and this also contradicts our assumption. Hence, **Control-Action** must **enable** σ.

– **Case 3, σ is skipped by agents 1 and 2:** The control-action procedures of both agents have executed "Response := SKIP" in Step 2.2. Hence, σ does not generate new behavior neither from $\mathrm{UR}^1_{\mathrm{ACT}^1}(\mathrm{NS}^1)$ in G^1 nor from $\mathrm{UR}^2_{\mathrm{ACT}^1}(\mathrm{NS}^2)$ in G^2. By the contrapositive of Property 5 and the induction hypothesis (ACT^1=$\mathrm{ACT}\cap\Sigma^1$, ACT^2=$\mathrm{ACT}\cap\Sigma^2$, NS^1=$\mathrm{NS}|^1$ and NS^2=$\mathrm{NS}|^2$) σ cannot generate new behavior from $\mathrm{UR}_{\mathrm{ACT}}(\mathrm{NS})$ in G. Hence, **Control-Action** skips σ at Step 2.2.

This completes the induction to prove Equality (ii).

Set of equalities **(iii)**: By Step 3 of **VLP-PO**, $\mathrm{PS}=\mathrm{UR}_{\mathrm{ACT}}(\mathrm{NS})$. Also,

$\mathrm{PS}^i = \mathrm{UR}^i_{\mathrm{ACT}^i}(\mathrm{NS}^i)$	[**DI-VLP-PO**^{i} Step 3].
$= \mathrm{UR}^i_{\mathrm{ACT}\cap\Sigma^i}(\mathrm{NS}|^i)$	[By the sets of equalities (i) and (ii) and the fact that the agents 1 and 2 take same action on each event.]
$= \mathrm{UR}_{\mathrm{ACT}}(\mathrm{NS})|^i$	[Property 1.]

This completes the proof of the basis of the main induction.

Induction Step (main induction): Assume the sets of equalities (i-iii) hold for all $s \in P(L(G, \gamma_p))$ of length n; we prove they also hold for $s\sigma$, for $\sigma \in \gamma_p(s)$.

Set of equalities **(i)**: By Step 3 of **VLP-PO**, NS=N_σ(PS). Also,

$$\begin{aligned} NS^i &= N^i_\sigma(PS^i) && \text{[DI-VLP-PO}^i \text{ Step 1.]} \\ &= N^i_\sigma(PS|^i) && \text{[induction hypothesis.]} \\ &= N_\sigma(PS)|^i && \text{[Property 3.]} \\ &= NS|^i. \end{aligned}$$

Equality **(ii)**: This is identical to the proof of Equality (ii) in the induction basis, with the following obvious changes: x_0 is replaced by some $x \in$NS such that $\delta(x_0, s') = x$ for some $s' = P(s)$ (the existence is clearly guaranteed) and t is replaced by $s't$.

Set of equalities **(iii)**: This proof is identical to proof of the set of equalities (iii) in the induction basis.

This completes the proof of the theorem. ■

Acknowledgements

The research of the first two authors was supported in part by the National Science Foundation under grant ECS-9057967 with additional support from DEC and GE. The research of the third author was supported in part by the National Science Foundation under grants ECS-9213922 and ECS-9315344. We wish to acknowledge several useful discussions with Karen Rudie, in particular on the topic of decentralized control.

Notes

1. The second policy of the paper.
2. It is assumed that this full observation control policy disables events only if necessary.
3. For $s \in P(L(G))$, $\gamma_\pi(s) \stackrel{\text{def}}{=} \bigcap_{s' \in P^{-1}(s)}(\gamma(s))$, where γ is the full observation control policy.
4. This means more than a single extended unobservable reach needs to be examined to compute the policy.

References

Barbeau, M., Custeau, G. and St-Denis, R. 1995. An algorithm for computing the mask value of the supremal normal sublanguage of a legal language. *IEEE Trans. Automatic Control* 40(4): 699–703.

Ben Hadj-Alouane, N., Lafortune, S. and Lin, F. 1993. Control of partially obseved discrete event systems with maximal variable lookahead policies. *Proceedings of the Thirty-first Annual Allerton Conference on Communications, Control, and Computing* pp. 898–907.

Ben Hadj-Alouane, N., Lafortune, S. and Lin, F. 1994a. A distributed on-line algorithm for supervisory control under partial observation. *Proc. 1994 Conf. Information Sciences and Systems* Princeton, NJ, March.

Ben Hadj-Alouane, N., Lafortune, S. and Lin, F. 1994b. Think globally, communicate, act locally: On-line parallel/distribued supervisory control. *Proc. 33rd IEEE Conf. on Decision and Control* Orlando, FL, pp. 3661–3666. December.

Ben Hadj-Alouane, N., Lafortune, S. and Lin, F. 1994c. Variable lookahead supervisory control with state information. *IEEE Trans. Automatic Control* 39(12): 2398–2410.

Brandt, R. D., Garg, V., Kumar, R., Lin, F., Marcus, S. I. and Wonham, W. M. 1990. Formulas for calculating supremal controllable and normal sublanguages. *Systems and Control Letters* 15(2): 111–117.

Cho, H. and Marcus, S. I. 1989a. On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation. *Math. Control Signals Systems* 2: 47–69.

Cho, H. and Marcus, S. I. 1989b. Supremal and maximal sublanguages arising in superior synthesis problems with partial observations. *Math. Systems Theory* 22: 177–211.

Chung, S. L., Lafortune, S. and Lin, F. 1992. Limited lookahead policies in supervisory control of discrete event systems. *IEEE Trans. Automatic Control* 37(12): 1921–1935.

Chung, S. L., Lafortune, S. and Lin, F. 1994. Supervisory control using variable lookahead policies. *Discrete Event Dynamic Systems: Theory and Applications* 4(3): 237–268.

Cieslak, R., Desclaux, C., Fawaz, A. and Varaiya, P. 1988. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. Automatic Control* 33(3): 249–260.

Garg, V. K. 1993. Parallel and distributed algorithms for supervisory control of discrete event systems. *Proc. 32th IEEE Conf. on Decision and Control* San Antonio, TX, pp. 2236–2241.

Heymann, M. and Lin, F. 1994. On-line control of partially observed discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications* 4(3): 221–236.

Inan, K. 1994. Nondeterministic supervision under partial observations. *Proc. 11th International Conference on Analysis and Optimization of Systems 1994*, pp. 39–48, Springer-Verlag, Lecture Notes in Control and Information Sciences, vol. 199.

Lin, F. and Wonham, W. M. 1988. On observability of discrete-event systems. *Information Sciences* 44: 173–198.

Ramadge, P. J. 1989. Some tractable supervisory control problems for discrete-event systems modeled by Buchi automata. *IEEE Trans. Automatic Control* 34(1): 10–19.

Ramadge, P. J. and Wonham, W. M. 1987. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization* 25(1): 206–230.

Ramadge, P. J. and Wonham, W. M. 1989. The control of discrete event systems. *Proc. IEEE* 77(1): 81–98.

Rudie, K. and Wonham, W. M. 1990. The infimal prefix-closed and observable superlanguage of a given language. *Systems and Control Letters* 15: 361–371.

Rudie, K. and Wonham, W. M. 1992. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. Automatic Control* 37(11): 1692–1708.

Wonham, W. M. and Ramadge, P. J. 1987. On the supremeal controllable sublanguage of a given language. *SIAM J. Control and Optimization* 25(3): 637–659.

Wonham, W. M. and Ramadge, P. J. 1988. Modular supervisory control of discrete event systems. *Math. Control Signals Systems* 1(1): 13–30.