Discrete-Event Control of Nondeterministic Systems

Michael Heymann and Feng Lin, *Member, IEEE*

***Abstract*—Nondeterminism in discrete-event systems occurs in many practical situations and often as a result of partial observability of events. For the adequate description of nondeterministic systems and nondeterministic phenomena, the trajectory-model formalism was introduced in [6] and [7]. This formalism has been used in [26] (also [14] and [15]) for obtaining various results on supervisory control of nondeterministic systems subject to language specifications. In the present paper we develop a theory of supervisory control for nondeterministic discrete-event systems subject to both language and trajectory-model specifications. We further show how well-known algorithms for supervisory control (of deterministic systems) under partial observation can be adapted for synthesis of supervisors for nondeterministic systems subject to both language and trajectory-model specifications.**

***Index Terms*—Discrete-event systems, nondeterminism, supervisory control, trajectory model.**

I. Introduction

MOST OF the published research on control of discrete-event systems (DES) has focused on systems that are modeled as deterministic finite state machines. For such systems, an extensive theory has been developed [24]. A great deal of attention was also given to the control of partially observed DES's [17], in which only a subset of the system's events are available for external observation. For such systems, necessary and sufficient conditions for existence of supervisors and algorithms for supervisor synthesis [3], [16]–[19], [23], [24] for off-line as well as on-line implementation [2], [8] have been obtained, and a wide variety of related questions have been investigated.

Partially observed systems frequently exhibit nondeterministic behavior. There are, however, situations in which the system's model is nondeterministic not because of partial observation but, rather, because either the system is inherently nondeterministic or because only a partial model of the system is available and some or all of its internal activities are unmodeled.

In contrast to deterministic DES's, whose behaviors are fully specified by their generated language, nondeterministic systems exhibit behaviors whose description requires much more refinement and detail. Further, while in the deterministic case, legal behavior of a system can be adequately expressed in terms of a language specification, this is clearly not always true when the system is nondeterministic. Indeed, to formally capture and specify legal behavior of the controlled system, it may be necessary to state, in addition to the permitted language, the degree of nondeterminism that the controlled system is allowed to retain. The *trajectory model* formalism was introduced in [6] and [7] as a semantic framework for modeling and specification of nondeterministic behaviors, and it was shown to adequately capture nondeterministic phenomena that one might wish to discriminate and distinguish by discrete-event control. Thus, for control purposes, nondeterministic DES's can be modeled either as nondeterministic automata (with ϵ-transitions) or as *trajectory models*.[1]

In recent years, there has been increasing interest in questions associated with nondeterminism in connection with supervisory control of DES's. In [4], [5], and [13], nondeterministic supervisors for DES's are considered, and existence conditions of supervisors are derived for various types of deterministic or nondeterministic specifications. In [15] and [26] the supervisory control problem is considered where the supervised system is assumed (or permitted) to be nondeterministic, while the specification is assumed to be deterministic (that is, a language specification) and the supervisor is also assumed to be deterministic. Conditions for supervisor existence are derived there, but no explicit algorithms for synthesis of supervisors are presented. In [14] nonblocking supervisory control of nondeterministic systems is considered where a concept of trajectory-model nonblocking (that differs from language-model nonblocking) is introduced. On the other hand, in [21] and [22], deterministic supervisors for nondeterministic plants with nondeterministic specifications are considered. They employ Hoare's *failures* semantics for system specification and derive certain algorithms (of high complexity) for supervisor synthesis. Indeed, it seems to be quite evident from the work reported in [21] and [22] that the direct supervisor synthesis for nondeterministic systems is quite a difficult task.

Motivated by the above, we began an investigation [9]–[11] of the connection between the supervisory control problem for general nondeterministic systems and the corresponding problem for partially observed deterministic systems. Our investigation led us to the conclusion that there appears to be neither a need, nor an advantage in developing a direct algorithmic approach to synthesis of supervisors for nondeterministic systems. In particular, we developed an approach to synthesis of supervisors for nondeterministic systems in which

Manuscript received February 2, 1996; revised December 4, 1996. Recommended by Associate Editor, E. K. P. Chong. This work was supported in part by the National Science Foundation under Grant ECS-9315344 and in part by the Technion Fund for Promotion of Research.

M. Heymann was with NASA Ames Research Center, Moffett Field, CA 94035 USA. He is now with the Department of Computer Science, Technion-Israel Institute of Technology, Haifa 32000, Israel.

F. Lin is with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202 USA.

Publisher Item Identifier S 0018-9286(98)01179-9.

[1] Other formalisms for modeling nondeterministic behaviors have been proposed in the literature, such as failures semantics and bisimulation semantics. These are briefly discussed in relation to discrete-event control in [7].

direct advantage is taken of the existing theory and algorithms for control under partial observation of deterministic systems.

Our approach to the supervisor synthesis is based on the following basic idea: we first synthesize from the given system, by adding to it hypothetical transitions and hypothetical uncontrollable and unobservable events, a deterministic system whose partially observed image (in the sense that the hypothetical events are obviously not observed) is the original nondeterministic system. We call this procedure *lifting*. Before performing the lifting, the legal (trajectory model) specification is embedded in the original nondeterministic system model in a way that can readily be dealt with in the corresponding lifted deterministic system. The next step of the synthesis is to construct a supervisor for the lifted system subject to the (obvious) condition that the artificially added events are neither observable nor controllable. Such a supervisor can readily be constructed using the well-known theory and algorithms for supervisory control of partially observed systems. It is self-evident, and we show it formally, that a supervisor synthesized in this way is applicable for the original nondeterministic system and satisfies the specifications. Moreover, we show that if the supervisor designed using this approach is optimal for the lifted system, it is also the optimal supervisor for the original system. Thus, since control under partial observation is well known, we only have to, ultimately, focus on the auxiliary steps of model lifting and specification embedding.

The simplest version of the supervisory control problem for nondeterministic systems is the case when the model is given as a nondeterministic automaton and the specification of legal behavior is given by a set of *illegal* states that must be avoided. This case, in which we refer to the specification as *static*, has been discussed in [10], and we review it here briefly for the sake of completeness. Basically, the only algorithmic step needed in the static case, prior to the employment of standard synthesis algorithms, is the lifting algorithm (which, as was shown in [10], can actually be sidestepped if one wishes to do so).

In the present paper we focus attention on the case where the specification is given as a trajectory model and where the central issue is the trajectory-embedding. That is, the main problem is the correct interpretation of the specification as a restriction of permitted system behavior. This is done by *embedding* of the specification in the plant model so that we can ultimately proceed, just as in the static case, using the lifting technique.

We deal in the present paper only with safety specifications and ignore the important question of liveness, or nonblocking, issues that are addressed extensively in [9].

In Section II, we briefly review the relevant aspects of the theory of supervisory control under partial observation, and in Section III we review the main concepts of nondeterministic DES's and their representations and reexamine the relation between the trajectory models and their corresponding nondeterministic automata. Also, a "lifting" formalism is presented by which the nondeterministic system is translated (or lifted) to a deterministic system, by introducing hypothetical events. The lifted system is constructed so that its projection yields the original nondeterministic system. In Section IV we discuss the supervisory control of nondeterministic systems with static specifications, in which the specification of legal behavior is given as a subset of legal states. In Section V we investigate in detail the problem of supervisory control with dynamic trajectory-model specifications. We develop an algorithmic framework for translation of the supervisory control problem with dynamic specifications to an equivalent problem with static specifications. Finally, in Section VI we conclude the paper with a brief discussion of the methodology for supervisor synthesis.

II. Deterministic Supervisory Control Under Partial Observation

In this section we briefly review the basic results of supervisory control for deterministic systems under partial observation. The uncontrolled system is described by an (deterministic) automaton $G = (\Sigma, Q, \delta, q_o, Q_m)$ with elements defined in a usual way. The languages generated and marked by G are denoted by $L(G)$ and $L_m(G)$, respectively. The event set is partitioned into controllable (observable) and uncontrollable (unobservable) disjoint subsets $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc} (= \Sigma_o \dot{\cup} \Sigma_{uo})$. A supervisor is a disablement map γ: $PL(G) \rightarrow 2^{\Sigma_c}$ (where P: $\Sigma^* \rightarrow \Sigma_o^*$ is the projection map that deletes the unobserved events) such that, following an observed string $s \in PL(G)$, $\gamma(s)$ denotes the set of events $\sigma \in \Sigma_c$ that are disabled by the supervisor. The languages generated by the supervised system is denoted by $L(\gamma/G)$ which is given inductively as follows.

1) $\epsilon \in L(\gamma/G)$.
2) $(\forall s \in L(\gamma/G))(\forall \sigma \in \Sigma) s\sigma \in L(\gamma/G) \Leftrightarrow s\sigma \in L(G) \wedge \sigma \notin \gamma(s)$.

We say that K is (prefix) closed if it equals its prefix closure $\overline{K}$. We also define controllability and observability as follows [17], [23].

Definition 1: A language $K \subseteq L(G)$ is said to be *controllable* (with respect to Σ_{uc}) if

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

Definition 2: A language $K \subseteq L(G)$ is said to be Δ-*observable* (with respect to $\Delta \subseteq \Sigma$ and Σ_{uo}), and when $\Delta = \Sigma$ simply *observable*, if

$$(\forall s, s' \in \overline{K}) Ps = Ps' \Rightarrow (\forall \sigma \in \Delta)(s\sigma \in \overline{K} \wedge s'\sigma \in L(G) \Rightarrow s'\sigma \in \overline{K}).$$

The controllability and observability characterize the existence condition for a supervisor as proved in [17].

Theorem 1: Let $K \subseteq L(G)$ be nonempty. Then there exists a supervisor γ such that $L(\gamma/G) = K$ if and only if K is closed, controllable, and observable.

Another useful concept is normality which is defined as follows [17].

Definition 3: A language $K \subseteq L(G)$ is *normal* if

$$(\forall s \in L(G)) Ps \in P\overline{K} \Rightarrow s \in \overline{K}.$$

One important fact regarding the relation between observability and normality is given by the following proposition, which is essential to the development in this paper [20].

Proposition 1: If $\Sigma_c \subseteq \Sigma_o$, then a language is controllable and observable if and only if it is controllable and normal.

A nice property of controllable (normal) languages is that they are closed under arbitrary union. So let $\mathcal{C}(E)$ ($\mathcal{N}(E)$, $\mathcal{CN}(E)$, respectively) denote the set of controllable (normal, controllable and normal, respectively) sublanguages of E, then we have the following [16], [17], [23].

Proposition 2: The supremal elements $\sup \mathcal{C}(E)$, $\sup \mathcal{N}(E)$, and $\sup \mathcal{CN}(E)$ exist, and they preserve the property of closedness.

III. Nondeterminism

In this section we briefly review the trajectory-model formalism of [7] (see also [26]) which has been developed as a basic tool for modeling and analysis of nondeterministic DES's.

Just as the *trace* $s \in L(G) \subset \Sigma^*$ is a record of the string of events executed in a given run of a system G, the *trajectory* is also a record associated with a run of G. It is more detailed than the trace in that it lists, in addition to the successfully executed events, events that the system might have rejected (or refused), if offered, after each successful event. Thus, a trajectory is an object in $2^\Sigma \times (\Sigma \times 2^\Sigma)^*$ of the form

$$t = (X_0, (\sigma_1, X_1), \cdots, (\sigma_k, X_k))$$

where σ_i denotes the ith executed event, and X_i, the ith *refusal*, denotes the set of events refused after the ith executed event. The *initial refusal* X_0 is the set of events that are refused before any event is executed. We call the integer k the *length* of t, denoted $|t|$, and the trace associated with t is defined as

$$\mathrm{tr}(t) = \sigma_1 \cdots \sigma_k.$$

A trajectory is called *valid* if $\sigma_i \notin X_{i-1}$ for all $i > 0$ (that is, an event cannot be executed if it has just been refused).

Let t be a trajectory given by

$$t = (X_0, (\sigma_1, X_1), \cdots, (\sigma_k, X_k)).$$

A trajectory r is a *prefix* of t, denoted $r \preceq t$, if

$$r = (X_0, (\sigma_1, X_1), \cdots, (\sigma_j, X_j))$$

and $0 \le j \le k$. The set of all prefixes of t is called the *prefix-closure* of t and is denoted $\mathrm{pref}(t)$.

A trajectory r is said to be *dominated* by t, denoted $r \sqsubseteq t$, if it is of the form

$$r = (Y_0, (\mu_1, Y_1), \cdots, (\mu_k, Y_k))$$

with $\mu_i = \sigma_i$ for $1 \le i \le k$ and $Y_j \subseteq X_j$ for $0 \le j \le k$. The set of all trajectories dominated by t is called the *completion*, or *dominance-closure*, of t and denoted $\mathrm{comp}(t)$.

Finally, we define the *closure* of t, denoted $\mathrm{cl}(t)$, as

$$\mathrm{cl}(t) := \bigcup_{v \in \mathrm{comp}(t)} \mathrm{pref}(v)$$

and the closure of a set of trajectories $\mathcal{T}$, is given by

$$\mathrm{cl}(\mathcal{T}) := \bigcup_{t \in \mathcal{T}} \mathrm{cl}(t).$$

A set of trajectories $\mathcal{T}$ is *closed*[2] if

$$\mathcal{T} = \mathrm{cl}(\mathcal{T}).$$

We say that a set of trajectories $\mathcal{T}$ is *saturated*[3] if the following condition holds:

$$\begin{aligned}&(\forall k = 1, 2, \cdots)(\forall j\colon 0 \le j \le k)(\forall \sigma \in \Sigma - X_j)\\ &\quad(((X_0, (\sigma_1, X_1), \cdots, (\sigma_k, X_k)) \in \mathcal{T}\\ &\quad\wedge (X_0, (\sigma_1, X_1), \cdots, (\sigma_j, X_j)(\sigma, \emptyset)) \notin \mathcal{T})\\ &\quad\Rightarrow (X_0, (\sigma_1, X_1), \cdots, (\sigma_j, X_j \cup \{\sigma\}) \cdots,\\ &\quad(\sigma_k, X_k)) \in \mathcal{T}).\end{aligned}$$

We are now in a position to define a (nondeterministic) process through its associated set of trajectories. Intuitively, we identify a process $\mathcal{P}$ with the set of all trajectories associated with possible runs of $\mathcal{P}$. More formally, we have the following.

Definition 4:[4] A (possibly) nondeterministic *process* $\mathcal{P}$ is a closed and saturated subset of valid trajectories $\mathcal{P} \subseteq 2^\Sigma \times (\Sigma \times 2^\Sigma)^*$.

The saturation condition on the set of trajectories of a process implies that if an event is impossible it will be refused. (We shall later see that while in nondeterministic processes events need not be impossible to be refused, in deterministic processes events are refused if and only if they are impossible.)

Let $\mathcal{T}$ be a set of trajectories. We say that a trajectory $t \in \mathcal{T}$ is *dominant* (in $\mathcal{T}$) if there is no trajectory $t' \in \mathcal{T}$, $t' \neq t$, such that $t \sqsubseteq t'$. The set of all trajectories that are dominant in $\mathcal{T}$ is called the *dominance-set* of $\mathcal{T}$ and is denoted $\mathrm{dom}(\mathcal{T})$.

The following proposition states that a process $\mathcal{P}$ is completely characterized by its dominance set [7] (see also [26, Th. 1]).

Proposition 3: Let $\mathcal{P}$ be a process. Then $\mathrm{cl}(\mathrm{dom}(\mathcal{P})) = \mathcal{P}$.

We shall next examine how trajectory-model representations of DES's, as defined above, are related to their more traditional representation as automata or state machines.

Let us consider a DES given by a nondeterministic finite automaton (possibly with ϵ-transitions)

$$\mathcal{P} = (\Sigma \cup \{\epsilon\}, Q, \delta, q_0)$$

over the event set Σ, with a nondeterministic transition function $\delta\colon Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$. Let us assume, further, that the system is nondivergent, that is, that there are no unbounded ϵ-paths (i.e., loops that consist of ϵ-transitions). To obtain the set of trajectories associated with $\mathcal{P}$, we proceed as follows. First, we associate with each state $q \in Q$ its *maximal-refusal-set* $X_q \subseteq \Sigma$, which is defined as

$$X_q := \{\sigma \in \Sigma\colon (\forall q' \in \epsilon^*(q))\delta(q', \sigma) = \emptyset\}$$

[2] A closed set of trajectories is always nonempty since it includes the null trajectory $(\emptyset, \epsilon)$.

[3] The term “saturated” as defined here differs from the way it was used in [26].

[4] The above definition is a simplified version of [7, Definition 12.1] since we deal here only with termination-free nondivergent processes. The concepts of termination and divergence are discussed in detail in [7]. Intuitively, a process is nondivergent if it cannot engage in an unbounded string of unobserved transitions.

where $\epsilon^*(q)$, the *ϵ-closure* of q, is defined inductively [12] as

$$q \in \epsilon^*(q) \text{ and } q' \in \epsilon^*(q) \Rightarrow \delta(q', \epsilon) \subseteq \epsilon^*(q).$$

With each path $p = (q_0, \sigma_1, q_1, \cdots, \sigma_k, q_k)$ in $\mathcal{P}$, we associate a trajectory t_p in the following way: first we represent p as a *formal* trajectory by replacing each state in p by its maximal refusal set. That is, we write $\tilde{t}_p := (X_{q_0}, \sigma_1, X_{q_1}, \cdots, \sigma_k, X_{q_k})$. (Note that in $\tilde{t}_p$, some of the σ_i's may be ϵ.) Then, to obtain the trajectory t_p associated with p, we delete all epsilons from $\tilde{t}_p$, and in the resulting string we replace all consecutive refusal sets by their union.

A state q is called *ϵ-stable* if $q = \epsilon^*(q)$, that is if $\delta(q, \epsilon) = \emptyset$. The assumption that $\mathcal{P}$ is nondivergent implies that, in a nontrivial process (that is, with a nonempty state set), there exists at least one ϵ-stable state in the ϵ-closure of each state.

Denoting the set of trajectories t_p associated with all ϵ-stable paths in $\mathcal{P}$ by $\mathrm{dom}(\mathcal{P})$ (a path $p = (q_0, \sigma_1, q_1, \cdots, \sigma_k, q_k)$ is *ϵ-stable* if q_k is ϵ-stable), the trajectory model of $\mathcal{P}$ (which we also denote $\mathcal{P}$) is obtained as $\mathcal{P} = \mathrm{cl}(\mathrm{dom}(\mathcal{P}))$.

Conversely, we recall [7] that we can construct a nondeterministic state machine (represented as a transition graph with ϵ transitions) directly from the set $\mathrm{dom}(\mathcal{P})$ or, more specifically, from the set $\mathcal{M}(\mathcal{P})$ defined as

$$\mathcal{M}(\mathcal{P}) := \bigcup_{t \in \mathrm{dom}(\mathcal{P})} \mathrm{pref}(t).$$

We identify the state set of the nondeterministic state machine with $\mathcal{M}(\mathcal{P})$ and construct the state-transition graph by induction on trajectory length as described in [7, Algorithm 12.1].[5] (See also the construction presented in [26] that uses so-called "saturated trajectories.")

It is not difficult to see that if $\mathcal{P}$ and $\mathcal{Q}$ are two processes such that $L(\mathcal{P}) = L(\mathcal{Q})$ ($L(\mathcal{P})$ denotes the trace set, or the language generated by $\mathcal{P}$), then we are justified in saying that $\mathcal{P}$ is *more nondeterministic* than $\mathcal{Q}$ whenever $\mathcal{Q} \subseteq \mathcal{P}$ (because $\mathcal{P}$ can be thought of as evolving into $\mathcal{Q}$ through ϵ-transitions [7]).

We can now define a *deterministic* process in the trajectory model setting.

Definition 5: A process $\mathcal{P}$ is called *deterministic* if for every trajectory $(X_0, (\sigma_1, X_1), \cdots, (\sigma_k, X_k)) \in \mathcal{P}$ and any $\sigma \in \Sigma$

$$\begin{aligned}&(X_0, (\sigma_1, X_1), \cdots, (\sigma_k, X_k), (\sigma, \emptyset)) \in \mathcal{P}\\ &\quad\Leftrightarrow (X_0, (\sigma_1, X_1), \cdots, (\sigma_k, X_k \cup \{\sigma\})) \notin \mathcal{P}.\end{aligned}$$

Thus, a process is deterministic whenever events are refused if and only if they are impossible. (Compare this condition with the saturation condition defined earlier.)

Now, let $L \subseteq \Sigma^*$ be a prefix-closed language (set of traces) and consider the set of all trajectory models that share L as their trace set. First, we need to convince ourselves that this set is never empty. To this end we shall construct a specific trajectory model in this set that we shall denote $\det(L)$ as follows:

```
Algorithm 1 (Construction of $\det(L)$)
  $(\emptyset, \epsilon) \in \det(L)$.
  Proceed by induction on string length:
  For $t = (X_0, (\sigma_1, X_1), \cdots, (\sigma_k, X_k)) \in \det(L)$
    and $\sigma \in \Sigma$,
      $(X_0, (\sigma_1, X_1), \cdots, (\sigma_k, X_k \cup \{\sigma\}))$
        $\in \det(L) \Leftrightarrow tr(t)^\frown\sigma \notin L$
      $(X_0, (\sigma_1, X_1), \cdots, (\sigma_k, X_k), (\sigma, \emptyset))$
        $\in \det(L) \Leftrightarrow \mathrm{tr}(t)^\frown\sigma \in L$.
```

The correctness of the above algorithm is stated in the following proposition [7, Proposition 12.5].

Proposition 4: $\det(L)$ is deterministic and $L(\det(L)) = L$.

The following theorem summarizes our preceding discussion and characterizes deterministic processes.

Theorem 2: Let $\mathcal{P}$ be a process, and let $L(\mathcal{P})$ be its trace set. Then $\mathcal{P}$ is deterministic if and only if for every process $\mathcal{Q}$ such that $L(\mathcal{Q}) = L(\mathcal{P})$

$$\mathcal{P} = \det(L(\mathcal{P})) \subseteq \mathcal{Q}.$$

Thus a deterministic process is uniquely defined by its associated trace set and, in fact, is the smallest process associated with a given trace set.

The validity of the following proposition [7, Th. 12.1] is easy to verify.

Proposition 5: The union of a nonempty set of processes is a process.

In view of the above proposition the union of the set of trajectory models that have $L \subseteq \Sigma^*$ as their trace set is also a trajectory model. It is of course the most nondeterministic trajectory model that has L as its trace set and is denoted $\mathrm{nondet}(L)$. It can easily be constructed from L as follows:

$$\mathrm{nondet}(L) = \mathrm{cl}\left(\bigcup_{s \in L} \bar{t}(s)\right)$$

where $\bar{t}(s) := (\Sigma - \{\sigma_1\}, \sigma_1, \cdots, \Sigma - \{\sigma_n\}, \sigma_n, \Sigma)$, for $s = \sigma_1\sigma_2\cdots\sigma_n$.

For each trajectory model, we can thus construct a corresponding nondeterministic automaton with ϵ-transitions, using the algorithm in [7]. Similarly, for each nondeterministic automaton with ϵ-transitions, we can construct its trajectory model by identifying each ϵ-stable state with a corresponding dominant trajectory as discussed earlier [7]. Therefore, we can use either of them to model a nondeterministic system. Henceforth in this paper, so long as no confusion arises, we shall use the same symbol to denote both the trajectory model and its associated nondeterministic automaton. The languages generated and marked by a nondeterministic automaton $\mathcal{P}$ are denoted by $L(\mathcal{P})$ and $L_m(\mathcal{P})$, respectively.

Consider a nondeterministic automaton, possibly with ϵ-transitions

$$\mathcal{R} = (\Sigma \cup \{\epsilon\}, Q, \delta, q_o)$$

[5] The transition-graph constructed there has a loop-free *tree* structure which is infinite even when a finite transition graph exists, unless $\mathrm{dom}(\mathcal{P})$ is finite. The question of finiteness of the system G that is associated with a process $\mathcal{P}$ is related to the concept of *regular* processes [7].

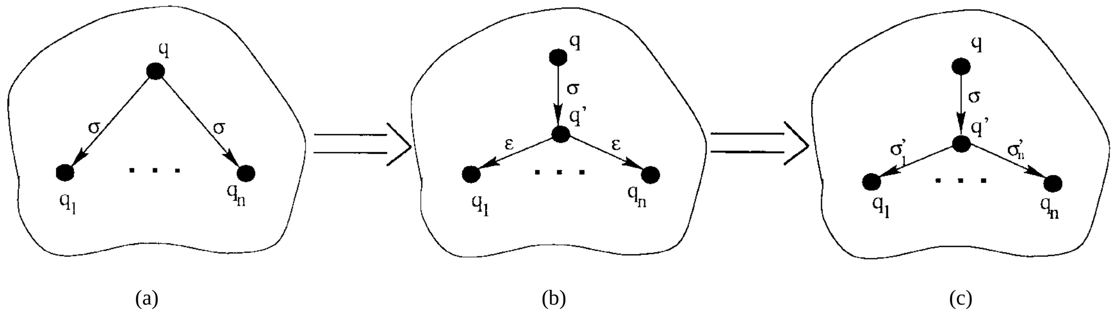


Fig. 1. Procedure Extend.

over the event set Σ. We introduce now a procedure for constructing a deterministic automaton

$$\tilde{\mathcal{R}} = (\Sigma \cup \Sigma', \tilde{Q}, \tilde{\delta}, q_o)$$

over an extended event set $\Sigma \cup \Sigma'$, such that $\mathcal{R} = \tilde{\mathcal{R}}\backslash_{\Sigma'}$. That is, $\tilde{\mathcal{R}}$ reduces to $\mathcal{R}$ upon replacing by ϵ-transitions all its transitions labeled by events in Σ'. The procedure is based on first extending $\mathcal{R}$ to a standard nondeterministic automaton with ϵ-transitions and then replacing the ϵ labels by labels from the event set $\Sigma' = \{\tau_1, \tau_2, \cdots\}$.

```
Procedure Extend ($\mathcal{R} \to \tilde{\mathcal{R}}$)
1.  $\tilde{Q} := Q$;
2.  For each $q \in \tilde{Q}$ and $\sigma \in \Sigma$
        If $|\delta(q, \sigma)| > 1$, add one more state, $q'$
        and add $\epsilon$-transitions as follows:
          $\tilde{Q} := \tilde{Q} \cup \{q'\}$;
          $\tilde{\delta}(q, \sigma) := \{q'\}$;
          $\tilde{\delta}(q', \epsilon) := \delta(q, \sigma)$;
    else set
          $\tilde{\delta}(q, \sigma) := \delta(q, \sigma)$;
3.  For each $q \in \tilde{Q}$
        replace the $\epsilon$-transitions by transitions
          labeled $\tau_1, \tau_2, \cdots$ as follows:
        If $\tilde{\delta}(q, \epsilon) = \{q_1, \cdots, q_n\}$, then set
          $\tilde{\delta}(q, \tau_1) := \{q_1\}$;
          $\cdots$
          $\tilde{\delta}(q, \tau_n) := \{q_n\}$;
4.  End of algorithm.
```

Using this procedure, we can "lift" a nondeterministic process to a deterministic process whose projection is the original nondeterministic process. This lifted process will be used in the rest of the paper.

Proposition 6: The lifted process $\tilde{\mathcal{R}}$ has the following properties.

1) The process $\tilde{\mathcal{R}}$ is deterministic.

2) $\mathcal{R} = \tilde{\mathcal{R}}\backslash_{\Sigma'}$.

Proof: That 1) holds is an elementary consequence of the construction.

To see that 2) holds, recall first that the trajectory model of a (nondeterministic) finite automaton is completely determined by its set of dominant trajectories, that is, by the set of trajectories associated with the ϵ-stable paths of the automaton. Thus, it will be sufficient to show that this set of trajectories is the same in $\mathcal{R}$ and in $\tilde{\mathcal{R}}\backslash_{\Sigma'}$. The procedure Extend performs two types of operations on $\mathcal{R}$, as illustrated in Fig. 1.

For the second type of operations, there is a one-to-one correspondence between paths of the form $(\cdots, q, \sigma, q', \epsilon, q_i, \cdots)$ in 1(b) and paths of the form $(\cdots, q, \sigma, q', \sigma_i', q_i, \cdots)$ in 1(c). Clearly, the projection of the latter yields the former. It remains to be shown that operations of the first type do not change the set of dominant trajectories. Indeed, by operations of the first type, paths of the form $(\cdots, q, \sigma, q_i, \cdots)$ are transformed to $(\cdots, q, \sigma, q', \epsilon, q_i, \cdots)$, $i = 1, \cdots, n$. The corresponding formal trajectories are of the form $(\cdots, X, \sigma, X_i, \cdots)$ and $(\cdots, X, \sigma, \bigcap_{j=1}^{n} X_j, \epsilon_i, X_i, \cdots)$, respectively, and it is readily noted that they yield identical trajectories after conclusion of the projection operation (since $(\bigcap_{j=1}^{n} X_j)\bigcup X_i = X_i$). ■

IV. Supervisory Control with Static Specifications

Clearly, nondeterministic systems exhibit more complex and more subtle behaviors than deterministic ones. It is not surprising that their behavioral specification can therefore also be more complex than that of deterministic systems. In the present section we examine the supervisory control problem of nondeterministic systems subject to very simple *static* (i.e., state-based) specifications wherein the system is restricted to remain within a predetermined subset of its state set. The more general case of *dynamic* specifications is discussed in Section V.

Suppose that the system under consideration is modeled as a nondeterministic automaton

$$\mathcal{P} = (\Sigma \cup \{\epsilon\}, Q, \delta, q_0)$$

and we are specified a subset $Q_b \subseteq Q$ of *forbidden* states that the system is not allowed to visit. Naturally, we assume that $q_0 \notin Q_b$. Control is achieved by a supervisor γ, defined as a function γ: $L(\mathcal{P}) \to 2^{\Sigma_c}$. Here, for $s \in L(\mathcal{P})$, $\gamma(s)$ is the set of (controllable) events that are disabled by the supervisor after execution of s. The *static* supervisory control problem is to construct a supervisor such that the supervised system satisfies the state restriction. To be more precise, let

$$\mathcal{P}_s = (\Sigma \cup \{\epsilon\}, Q_s, \delta_s, q_0)$$

be the restriction of the automaton $\mathcal{P}$ to the subset of "good" states $Q_s = Q - Q_b$; where δ_s: $Q_s \times \Sigma \to 2^{Q_s}$ is defined as $\delta|_{Q_s}$, that is

$$\delta_s(q, \sigma) := \delta(q, \sigma) \cap Q_s.$$

Our task is to synthesize a supervisor γ such that the supervised systems satisfies

$$\gamma/\mathcal{P} = \mathcal{P}_s.$$

We will derive an existence condition for such a supervisor. When this condition cannot be satisfied, we will synthesize a minimally restrictive supervisor that confines the supervised system to good states Q_s. We will discuss the synthesis of the minimally restrictive supervisor in Section VI.

In the above, the supervised system, denoted by $\gamma/\mathcal{P}$, is formally obtained as follows. First, the language $L(\gamma/\mathcal{P})$, generated by $\gamma/\mathcal{P}$, is given inductively as

1) $\epsilon \in L(\gamma/\mathcal{P})$;
2) $(\forall s \in L(\gamma/\mathcal{P}))(\forall \sigma \in \Sigma) s\sigma \in L(\gamma/\mathcal{P}) \Leftrightarrow s\sigma \in L(\mathcal{P}) \wedge \sigma \notin \gamma(s)$.

Next, to obtain the trajectory model of the supervised system, we recall [7] that the strict synchronous (parallel) composition $\mathcal{P}||\mathcal{T}(= \mathcal{P}_\Sigma||_\Sigma\mathcal{T})$ of two trajectory models, $\mathcal{P}$ and $\mathcal{T}$, is given as follows.[6] A trajectory

$$(Z_0, \sigma_1, Z_1, \cdots, \sigma_k, Z_k) \in \mathcal{P}||\mathcal{T}$$

if and only if $\sigma_1 \cdots \sigma_k \in L(\mathcal{P}) \cap L(\mathcal{T})$ and there exist dominant trajectories

$$(X_0, \sigma_1, X_1, \cdots, \sigma_k, X_k) \in \mathcal{P}$$

and

$$(Y_0, \sigma_1, Y_1, \cdots, \sigma_k, Y_k) \in \mathcal{T}$$

such that $Z_i \subseteq X_i \cup Y_i$ for all i: $0 \le i \le k$.

We can now prove the following.

Lemma 1: The trajectory model $\gamma/\mathcal{P}$ of the supervised system is given by

$$\gamma/\mathcal{P} = \mathcal{P}|| \operatorname{det}(L(\gamma/\mathcal{P})).$$

Proof: After executing the strings ϵ, $(\sigma_1), \cdots, s_k = (\sigma_1 \cdots \sigma_k)$, the supervisor disables (refuses) events in $\gamma(\epsilon)$, $\gamma(\sigma_1), \cdots, \gamma(s_k)$, respectively. Therefore, the dominant trajectories of $\gamma/\mathcal{P}$ are of the form

$$(X_0 \cup \gamma(\epsilon), \sigma_1, X_1 \cup \gamma(\sigma_1), \cdots, \sigma_k, X_k \cup \gamma(s))$$

where $(X_0, \sigma_1, X_1, \cdots, \sigma_k, X_k) \in \mathcal{P}$ is a dominant trajectory, and $\sigma_j \notin \gamma(s_{j-1})$ for $j = 1, \cdots, k$.

For $t \in L(\gamma/\mathcal{P})$, denote

$$\begin{aligned}\Sigma(t) &= \{\sigma \in \Sigma \colon t\sigma \notin L(\gamma/\mathcal{P})\}\\ &= \{\sigma \in \Sigma \colon t\sigma \notin L(\mathcal{P}) \vee \sigma \in \gamma(t)\}\\ &= \{\sigma \in \Sigma \colon t\sigma \notin L(\mathcal{P})\} \cup \gamma(t).\end{aligned}$$

By Algorithm 1, the dominant trajectories of $\operatorname{det}(L(\gamma/\mathcal{P}))$ are of the form

$$(\Sigma(\epsilon), \sigma_1, \Sigma(\sigma_1), \cdots, \sigma_k, \Sigma(s))$$

where $s = \sigma_1 \cdots \sigma_k \in L(\gamma/\mathcal{P})$. Therefore, a trajectory

$$(Z_0, \sigma_1, Z_1, \cdots, \sigma_k, Z_k) \in \mathcal{P}|| \operatorname{det}(L(\gamma/\mathcal{P}))$$

if and only if $\sigma_1 \cdots \sigma_k \in L(\mathcal{P}) \cap L(\gamma/\mathcal{P}) = L(\gamma/\mathcal{P})$, and there exist dominant trajectories

$$(X_0, \sigma_1, X_1, \cdots, \sigma_k, X_k) \in \mathcal{P}$$

and

$$(Y_0, \sigma_1, Y_1, \cdots, \sigma_k, Y_k) \in \operatorname{det}(L(\gamma/\mathcal{P}))$$

such that for all $s_i = \sigma_1 \cdots \sigma_i$, $0 \le i \le k$

$$\begin{aligned}Z_i &\subseteq X_i \cup Y_i\\ &= X_i \cup \{\sigma \in \Sigma \colon s_i\sigma \notin L(\mathcal{P})\} \cup \gamma(s_i)\\ &= X_i \cup \gamma(s_i).\end{aligned}$$

Comparing this trajectory model with $\gamma/\mathcal{P}$, it is clear that

$$\gamma/\mathcal{P} = \mathcal{P}|| \operatorname{det}(L(\gamma/\mathcal{P})).$$

■

To proceed with our analysis, it is convenient to first embed the specification $\mathcal{P}_s$ in the process $\mathcal{P}$ by considering the automaton

$$\overline{\mathcal{P}} = (\Sigma \cup \{\epsilon\}, Q, \delta, q_0, Q_b)$$

where the specification is interpreted as the subautomaton obtained by the restriction of $\overline{\mathcal{P}}$ to $Q_s = Q - Q_b$. Now we *lift* $\overline{\mathcal{P}}$ by applying to it the procedure Extend to obtain the deterministic automaton

$$\tilde{\mathcal{P}} = (\Sigma \cup \Sigma', \tilde{Q}, \tilde{\delta}, q_0, \tilde{Q}_b)$$

where

$$\tilde{Q}_b = Q_b \cup \{q \in \tilde{Q} - Q \colon (\forall t)\tilde{\delta}(q, t) \in Q_b\}.$$

The "legal" language $E \subseteq L(\tilde{\mathcal{P}})$ is now defined to be the set of all strings that visit only good states in $\tilde{\mathcal{P}}$, that is

$$E = \{s \in L(\tilde{\mathcal{P}}) \colon (\forall t \le s)\tilde{\delta}(q_o, t) \notin \tilde{Q}_b\}.$$

In view of Proposition 6, it is clear that

$$\tilde{\mathcal{P}}\backslash_{\Sigma'} = \mathcal{P}$$

and it is not difficult to prove the following.

Proposition 7:

$$\operatorname{det}(E)\backslash_{\Sigma'} = \mathcal{P}_s.$$

[6] This is a special case of the more general definition given in [7].

Proof: Denote

$$\tilde{\mathcal{P}}_s = (\Sigma \cup \Sigma', \tilde{Q} - \tilde{Q}_b, \tilde{\delta}|_{\tilde{Q}-\tilde{Q}_b}, q_o)$$

and

$$\mathcal{P}_s = (\Sigma \cup \{\epsilon\}, Q - Q_b, \delta|_{Q-Q_b}, q_o).$$

Then, clearly, $\tilde{\mathcal{P}}_s$ is deterministic, and $L(\tilde{\mathcal{P}}_s) = E$. Therefore, if we can show that $\tilde{\mathcal{P}}_s$ is a process obtained by the lifting of $\mathcal{P}_s$ (i.e., a lifted process of $\mathcal{P}_s$), then, by Proposition 6

$$\det(E)\backslash_{\Sigma'} = \tilde{\mathcal{P}}_s\backslash_{\Sigma'} = \mathcal{P}_s.$$

To show that $\tilde{\mathcal{P}}_s$ is a lifted process of $\mathcal{P}_s$, we note that $\tilde{\mathcal{P}}_s$ is a subautomaton of $\tilde{\mathcal{P}}$, which, in turn, is a lifted process of $\overline{\mathcal{P}}$. We further note that $\mathcal{P}_s$ is a subautomaton of $\overline{\mathcal{P}}$. Therefore, we only need to show that a path in $\overline{\mathcal{P}}$ visits a state in Q_b if and only if the corresponding lifted path in $\tilde{\mathcal{P}}$ visits a state in $\tilde{Q}_b$. (A path $\tilde{p}$ in $\tilde{\mathcal{P}}$ is the corresponding path of p in $\overline{\mathcal{P}}$, if it reduces to p after deletion from it of all states in $\tilde{Q} - Q$ and all events in Σ'. This implies, in particular, that the last state of $\tilde{p}$ is in Q.)

Only If: Assume that a path $p = (\cdots, q_{i-1}, \sigma_i, q_i, \cdots, \sigma_j, q_j, \cdots)$ of $\overline{\mathcal{P}}$ visits a bad state $q_i = q_b \in Q_b$. The corresponding path in $\tilde{\mathcal{P}}$ has the same form with possible insertions of σ', q', where $\sigma' \in \Sigma'$ and $q' \in \tilde{Q} - Q$. Hence the corresponding path in $\tilde{\mathcal{P}}$ also visits $q_b \in Q_b \subseteq \tilde{Q}_b$.

If: Consider a path in $\tilde{\mathcal{P}}$ of the form $(\cdots, q_{i-1}, \sigma_i, q_i, \cdots, \sigma_j, q_j, \cdots, q_k)$ that visits a state $q_b \in \tilde{Q}_b$. If $q_b \in Q_b$, then the projected path in $\overline{\mathcal{P}}$ also visits the state q_b. If $q_b \in \tilde{Q}_b - Q_b$, then $q_b \neq q_k$ and by the definition of $\tilde{Q}_b$, the next state visited by the path in $\tilde{\mathcal{P}}$ must be in Q_b. This bad state will be visited also by the projected path in $\overline{\mathcal{P}}$. ■

We can now prove the following theorem that provides the theoretical justification for our proposed approach to control of nondeterministic systems. Specifically, we shall state a necessary and sufficient condition for the existence of a supervisor. In the following theorem, observability is defined for $\Delta = \Sigma \cup \Sigma'$ and $\Sigma_{uo} = \Sigma'$, and the set of controllable events is $(\Sigma \cup \Sigma')_c = \Sigma_c$.

Theorem 3: For $\mathcal{P}$ and $\mathcal{P}_s$ given as above, there exists a supervisor γ such that $\gamma/\mathcal{P} = \mathcal{P}_s$ if and only if E is controllable and observable with respect to $L(\tilde{\mathcal{P}})$.

Proof: By the results of [17], E is controllable and observable with respect to $L(\tilde{\mathcal{P}})$ if and only if there exists a supervisor $\tilde{\gamma}$: $PL(\tilde{\mathcal{P}}) \rightarrow 2^{(\Sigma\cup\Sigma')_c}$ such that $L(\tilde{\gamma}/\tilde{\mathcal{P}}) = E$, where P: $(\Sigma \cup \Sigma')^* \rightarrow \Sigma^*$ is the projection map. Therefore, the proof of the theorem reduces to showing that there exists a supervisor γ such that $\gamma/\mathcal{P} = \mathcal{P}_s$ if and only if there exists a supervisor $\tilde{\gamma}$ such that $L(\tilde{\gamma}/\tilde{\mathcal{P}}) = E$.

If: Suppose that there exists a supervisor $\tilde{\gamma}$: $PL(\tilde{\mathcal{P}}) \rightarrow 2^{(\Sigma\cup\Sigma')_c}$ such that $L(\tilde{\gamma}/\tilde{\mathcal{P}}) = E$. Since $PL(\tilde{\mathcal{P}}) = L(\mathcal{P})$, and since $(\Sigma \cup \Sigma')_c = \Sigma_c$, we can define γ: $L(\mathcal{P}) \rightarrow 2^{\Sigma_c}$ as $\gamma(s) = \tilde{\gamma}(s)$. Thus we obtain

$$\gamma/\mathcal{P} = (\tilde{\gamma}/\tilde{\mathcal{P}})\backslash_{\Sigma'} = \det(E)\backslash_{\Sigma'} = \mathcal{P}_s$$

where the first equality above follows from Proposition 6 and the third equality follows from Proposition 7.

Only If: Suppose that there exists a supervisor γ: $L(\mathcal{P}) \rightarrow 2^{\Sigma_c}$ such that $\gamma/\mathcal{P} = \mathcal{P}_s$, and define $\tilde{\gamma}$: $PL(\tilde{\mathcal{P}}) \rightarrow 2^{(\Sigma\cup\Sigma')_c}$ as $\tilde{\gamma}(s) = \gamma(s)$. Thus, we have (with the aid of Propositions 6 and 7)

$$\det(E)\backslash_{\Sigma'} = \mathcal{P}_s = \gamma/\mathcal{P} = \tilde{\gamma}/\mathcal{P} = (\tilde{\gamma}/\tilde{\mathcal{P}})\backslash_{\Sigma'}$$

which, in view of the definition of E, implies that

$$L(\tilde{\gamma}/\tilde{\mathcal{P}}) = E.$$

■

The above theorem shows that we can translate a supervisory control problem of a nondeterministic system, subjected to static specifications, into a supervisory control problem under partial observation of a lifted deterministic system. The supervisors for both systems are the same ($\gamma = \tilde{\gamma}$). In the next section we shall show how the same approach can also be employed in the more complex setting of dynamic specifications. Later, we shall turn to the algorithmic aspects of supervisor synthesis.

V. Supervisory Control with Dynamic Specifications

In this section we again assume that the system under consideration is a nondeterministic automaton

$$\mathcal{P} = (\Sigma \cup \{\epsilon\}, Q, \delta, q_0)$$

but the specification of legal behavior is a *dynamic specification*, given to us as another (generally nondeterministic) automaton

$$\mathcal{H} = (\Sigma \cup \{\epsilon\}, H, \psi, h_0).$$

This nondeterministic specification automaton $\mathcal{H}$ is constructed so as to capture both the language constraints for the controlled system and, more subtly, the nondeterministic behaviors that the controlled system is allowed to retain. We shall see later, through an example, some typical nondeterministic control considerations.

Our goal is to design a supervisor γ (if possible) such that

$$\gamma/\mathcal{P} = \mathcal{H}.$$

For this to be possible, a precondition is that all the trajectories in $\mathcal{H}$ be physically possible in some subautomaton of $\mathcal{P}$. If this precondition is not satisfied, then $\mathcal{H}$ must be modified.

This is similar to the case of deterministic systems with language specifications, where in order for a supervisor γ to exist such that $L(\gamma/G) = E$, the precondition is $E \subseteq L(G)$. If this precondition is not satisfied, E is modified by replacing it with $E \cap L(G)$.

For nondeterministic systems, however, the situation is much more complex. We cannot simply modify $\mathcal{H}$ by taking $\mathcal{H} \cap \mathcal{P}$ since, in particular, the intersection of two processes is generally not even a process.

In this section, we will develop a procedure that will modify $\mathcal{H}$ correctly and, at the same time, translate the dynamic specification into an equivalent static specification.

As the first step in our procedure, we note that a trajectory in $\mathcal{H}$ whose trace is not in $L(\mathcal{P})$ is definitely impossible in $\mathcal{P}$ (or any of its subautomata). Therefore, we first replace $\mathcal{H}$ by

$$\hat{\mathcal{H}} = (\Sigma \cup \{\epsilon\}, \hat{H}, \hat{\psi}, \hat{h}_0) := \mathcal{H} || \operatorname{det}(L(\mathcal{P})).$$

It is not difficult to see that $\hat{\mathcal{H}}$ satisfies the constraint

$$L(\hat{\mathcal{H}}) = L(\mathcal{H}) \cap L(\mathcal{P}) \subseteq L(\mathcal{P})$$

and retains all the relevant nondeterministic aspects of $\mathcal{H}$.

Next, we note that $\hat{\mathcal{H}}$ imposes both a language constraint

$$L(\gamma/\mathcal{P}) = L(\hat{\mathcal{H}})$$

and a trajectory-model constraint[7]

$$\gamma/\mathcal{P} = \hat{\mathcal{H}}.$$

To consider the language constraints imposed by the specification, we construct the deterministic automaton

$$\hat{\mathcal{H}}_d = (\Sigma, \hat{H}_d, \hat{\psi}_d, \hat{h}_{d0}) := \operatorname{det}(L(\hat{\mathcal{H}}))$$

which we shall employ for our language specification. To this end, we extend $\hat{\mathcal{H}}_d$ to an automaton

$$\overline{\mathcal{H}}_d = (\Sigma, \overline{H}_d, \overline{\psi}_d, \hat{h}_{d0}, \hat{H}_d)$$

where $\overline{H}_d := \hat{H}_d \cup \{h_b\}$, where h_b is a new state that we call the *bad* state, and $\overline{\psi}_d$ is defined as

$$\overline{\psi}_d(\hat{h}_d, \sigma) = \begin{cases} \hat{\psi}_d(\hat{h}_d, \sigma), & \text{if } \hat{\psi}_d(\hat{h}_d, \sigma) \text{ is defined} \\ h_b, & \text{otherwise.} \end{cases}$$

We immediately note that $L(\overline{\mathcal{H}}_d) = \Sigma^*$ and $L_m(\overline{\mathcal{H}}_d) = L(\hat{\mathcal{H}}_d) = L(\hat{\mathcal{H}})$.

Next we construct the automaton $\overline{\mathcal{P}} = \mathcal{P} || \overline{\mathcal{H}}_d$, which can be represented as

$$\overline{\mathcal{P}} = (\Sigma \cup \{\epsilon\}, \overline{Q}, \overline{\delta}, \overline{q}_0, \overline{Q}_g).$$

We can now readily prove that the trajectory models of $\overline{\mathcal{P}}$ and of $\mathcal{P}$ coincide.

Proposition 8:

$$\overline{\mathcal{P}} = \mathcal{P}.$$

Proof: The result is an immediate consequence of the fact that, since $\overline{\mathcal{H}}_d$ is deterministic and $L(\overline{\mathcal{H}}_d) = \Sigma^*$, a trajectory t is in $\overline{\mathcal{H}}_d$ if and only if it is of the form $t = (\emptyset, (\sigma_1, \emptyset), \cdots, (\sigma_k, \emptyset))$ (see Definition 5 and the discussion preceding Lemma 1). ■

We also note at once that

$$L_m(\overline{\mathcal{P}}) = L(\mathcal{P}) \cap L_m(\overline{\mathcal{H}}_d) = L(\hat{\mathcal{H}}).$$

Thus, the problem of synthesizing a supervisor γ that maximizes $L(\gamma/\mathcal{P})$ subject to the constraint that $L(\gamma/\mathcal{P}) \subseteq L(\mathcal{H})$ is equivalent to synthesizing a supervisor $\overline{\gamma}$ that maximizes $L(\overline{\gamma}/\overline{\mathcal{P}})$ subject to the constraint that $L(\overline{\gamma}/\overline{\mathcal{P}}) \subseteq L_m(\overline{\mathcal{P}})$. This latter problem consists of synthesizing a minimally restrictive supervisor such that all paths of $\overline{\gamma}/\overline{\mathcal{P}}$ are confined to the subset of good states $\overline{Q}_g = Q \times \hat{H}_d$. This is clearly a supervisory control problem with static specifications of the type discussed in Section IV.

We now turn to the more restrictive aspect of our specification, namely to the requirement that the supervised system satisfy the trajectory-model specification $\gamma/\mathcal{P} = \hat{\mathcal{H}}$.

However, before addressing the technical aspects of this problem, it is in order to make a few observations regarding the relation between the language specification and the trajectory-model specification.

The language specification admits as "legal," every trajectory of the controlled system, so long as the associated trace is an element of the specified language. Thus, every trajectory-model $\mathcal{T}$ that satisfies the condition that $L(\mathcal{T}) = L(\hat{\mathcal{H}})$ will yield the same controlled system, provided only the language constraint is employed. The largest such trajectory model is $\operatorname{nondet}(L(\hat{\mathcal{H}}))$, which is obtained as the union of all trajectory models that share this language. Thus, we may think of the language specification as a trajectory-model specification with respect to the trajectory model $\operatorname{nondet}(L(\hat{\mathcal{H}}))$. Since this trajectory model is the most nondeterministic in its class, it is clear that the language specification does not discriminate between nondeterministic aspects of system behavior. It is therefore the role of the trajectory-model specification to delineate the nondeterministic behaviors that the controlled plant is permitted to retain.

Again, in view of Proposition 8, we shall employ $\overline{\mathcal{P}}$ as our plant model. Indeed, in this model we already marked the set $\overline{Q}_g$ of all the "good" states such that $L_m(\overline{\mathcal{P}}) = L(\hat{\mathcal{H}})$. It remains now only to determine the subset of these "good" states that consists of all states that can be reached via paths whose associated trajectories are in $\hat{\mathcal{H}}$. (The trajectory associated with a path has been defined in Section III.) More precisely, we wish to construct from $\overline{\mathcal{P}}$, the automaton

$$\overline{\mathcal{P}}_t = (\Sigma \cup \{\epsilon\}, \overline{Q}, \overline{\delta}, \overline{q}_0, \overline{Q}_t)$$

such that a path of $\overline{\mathcal{P}}_t$

$$p = (\overline{q}_0, \epsilon, \overline{q}_0^2, \cdots, \epsilon, \overline{q}_0^{i_0}, \sigma_1, \overline{q}_1^1, \cdots, \sigma_k, \overline{q}_k^1, \epsilon, \cdots, \overline{q}_k^{i_k})$$

belongs to $\overline{Q}_t$ (in the sense that each of its states belongs to $\overline{Q}_t$) if and only if its associated trajectory $t_p \in \hat{\mathcal{H}}$. Thus, $\overline{Q}_t$ is the largest subset of states in $\overline{Q}$ that can be reached by paths in $\overline{\mathcal{P}}_t$, whose associated trajectories are in $\hat{\mathcal{H}}$. To this end we employ the following algorithm that identifies in the

[7] We shall later see that when this specification cannot be met exactly, we will be able to obtain its best approximation.

process all paths whose associated trajectories are dominated by (corresponding) trajectories of the specification

```
Algorithm 2 (Trajectory inclusion)
Input:
  • Plant automaton: $\overline{\mathcal{P}} = (\Sigma \cup \{\epsilon\}, \overline{Q}, \overline{\delta}, \overline{q}_0, \overline{Q}_g)$
  • Specification automaton: $\hat{\mathcal{H}} = (\Sigma \cup \{\epsilon\}, \hat{H}, \hat{\psi}, \hat{h}_0)$.
satisfying $L(\hat{\mathcal{H}}) \subseteq (L(\overline{\mathcal{P}}))$.
Output:
  • Automaton: $\overline{\mathcal{P}}_t = (\Sigma \cup \{\epsilon\}, \overline{Q}, \overline{\delta}, \overline{q}_0, \overline{Q}_t)$.

Preliminaries
  • Represent $\overline{\mathcal{P}}$ and $\hat{\mathcal{H}}$ as trajectory model automata
    by augmenting each state label $r$ with its maximal
    refusal set $X_r$.
  • Set $\overline{M} := \overline{Q}_g$.
  • Set $M := \emptyset$.
  • Set $\overline{Q}_t := \emptyset$.
  • Set $(\forall \overline{q} \in \overline{Q}_g) R(\overline{q}) := \emptyset$.

Initialize algorithm
   1. Set $T := \epsilon^*(\overline{q}_0) \cap \overline{M}$. If $T \neq \emptyset$, set $\overline{q} := \overline{q}_0$.
      If $T = \emptyset$, go to End.
   2. Set $E := \epsilon^*(\hat{h}_0)$.
   3. Choose a state $\hat{h} \in E$.
   4. If $X_{\overline{q}} \subseteq X_{\hat{h}}$, add $\hat{h}$ to $R(\overline{q})$.
   5. Remove $\hat{h}$ from $E$.
   6. If $E \neq \emptyset$, go to 3.
   7. If $R(\overline{q}) \neq \emptyset$, add $\overline{q}$ to $\overline{Q}_t$ and to $M$.
   8. Remove $\overline{q}$ from $\overline{M}$ and from $T$.
   9. If $T \neq \emptyset$, choose a state $\overline{q} \in T$ and go to 2.

Iterate
  10. If $M = \emptyset$ go to End.
  11. Choose a state $\overline{q} \in M$ and set
            $S = \Sigma_q := \{\sigma \in \Sigma: |\overline{\delta}(\overline{q}, \sigma)| > 0\}$.
  12. If $S = \emptyset$, remove $\overline{q}$ from $M$ and go to 10.
  13. Choose a symbol $\sigma \in S$ and set
            $T := \epsilon^*(\overline{\delta}(\overline{q}, \sigma)) \cap \overline{M}$
               $= \{\epsilon^*(\overline{q}'): \overline{q}' \in \overline{\delta}(\overline{q}, \sigma)\} \cap \overline{M}$,
      and
            $E(\overline{q}, \sigma) := \cup_{\hat{h} \in R(\overline{q})} \{\epsilon^*(\hat{\psi}(\hat{h}, \sigma))\}$.
  14. If $T = \emptyset$, remove $\sigma$ from $S$ and go to 12.
  15. Choose a state $\overline{q}' \in T$ and set $E = E(\overline{q}, \sigma)$.
  16. Choose a state $\hat{h}' \in E$.
  17. If $X_{\overline{q}'} \subseteq X_{\hat{h}'}$, add $\hat{h}'$ to $R(\overline{q}')$.
  18. Remove $\hat{h}'$ from $E$.
  19. If $E \neq \emptyset$ go to 16.
  20. If $R(\overline{q}') \neq \emptyset$, add $\overline{q}'$ to $\overline{Q}_t$ and to $M$.
  21. Remove $\overline{q}'$ from $\overline{M}$ and from $T$ and go to 14.

End. ■
```

The correctness of Algorithm 2 is stated in the following.

Theorem 4: For any path p in $\overline{\mathcal{P}}_t$, p belongs to $\overline{Q}_t$ if and only if $t_p \in \hat{\mathcal{H}}$.

Outline of Proof: We only give an outline of the proof because its details are tedious and provide no additional insight.

First, we note from the algorithm that a state $\overline{q} \in \overline{Q}_t$ if and only if $R(\overline{q}) \neq \emptyset$.

Next, it is not difficult to prove that a state $\hat{h} \in \hat{H}$ satisfies $\hat{h} \in R(\overline{q})$ if and only if for every path

$$p = (\overline{q}_0, \epsilon, \overline{q}_0^2, \cdots, \epsilon, \overline{q}_0^{i_0}, \sigma_1, \overline{q}_1^1, \cdots, \sigma_k, \overline{q}_k^1, \epsilon, \cdots, \overline{q})$$

leading to $\overline{q}$ and belonging to $\overline{Q}_t$, there exists a path in $\hat{\mathcal{H}}$

$$\hat{p} = (\hat{h}_0, \epsilon, \hat{h}_0^2, \cdots, \epsilon, \hat{h}_0^{j_0}, \sigma_1, \hat{h}_1^1, \cdots, \sigma_k, \hat{h}_k^1, \epsilon, \cdots, \hat{h})$$

such that $\mathrm{tr}(p) = \mathrm{tr}(\hat{p}) = \sigma_1 \cdots \sigma_k$, and

$$(\forall n, 0 \leq n \leq k) \quad X_{\overline{q}_n^{i_n}} \subseteq X_{\hat{h}_n^{j_n}}.$$

Hence, a path

$$p = (\overline{q}_0, \epsilon, \overline{q}_0^2, \cdots, \epsilon, \overline{q}_0^{i_0}, \sigma_1, \overline{q}_1^1, \cdots, \sigma_k, \overline{q}_k^1, \epsilon, \cdots, \overline{q}_k^{i_k})$$

belongs to $\overline{Q}_t$ if and only if there exists a path in $\hat{\mathcal{H}}$

$$\hat{p} = (\hat{h}_0, \epsilon, \hat{h}_0^2, \cdots, \epsilon, \hat{h}_0^{j_0}, \sigma_1, \hat{h}_1^1, \cdots, \sigma_k, \hat{h}_k^1, \epsilon, \cdots, \hat{h}_k^{j_k})$$

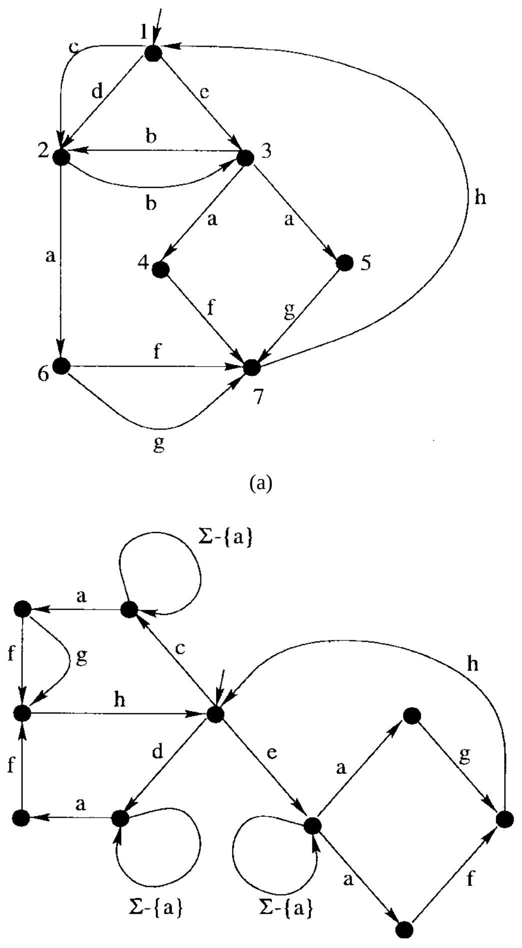
such that

$$(\forall n, 0 \leq n \leq k) \quad X_{\overline{q}_n^{i_n}} \subseteq X_{\hat{h}_n^{j_n}}.$$

But this implies that $t_p \in \mathrm{comp}(t_{\hat{p}}) \subset \hat{H}$, concluding the proof. ■

The above theorem shows that we can always translate a dynamic specification into an equivalent static specification. Next we give an example to illustrate the preceding theory.

Example 1: The process $\mathcal{P}$ in Fig. 2(a) represents, schematically, a message transmission system that sends messages from a source (state 1) to a destination (state 7). The system has at its disposal two terminals (represented by states 2 and 3) through which messages can be forwarded for transmission (event a). Two communication channels are available for message transmission: a secure channel and a nonsecure channel. Transmission on the secure channel is denoted by event f and on the nonsecure channel by event g. Upon completion of successful transmission of a message, an acknowledgment is sent from the destination to the source (event h), thereby permitting transmission of a new message. Three types of messages can be sent in the system: top-secret messages that are initially dispatched to terminal 2 (event d), secret messages that are also initially dispatched to terminal 2 (event c), and nonsecret messages that are initially dispatched to terminal 3 (event e). Messages can be transferred between the terminals prior to their transmission (event b). When a message is forwarded for transmission (event a) from terminal 2, it can be transmitted controllably (at the discretion of the sender) on either the secure or on the nonsecure channel. On the other hand, when a message is forwarded from terminal 3,



(a)

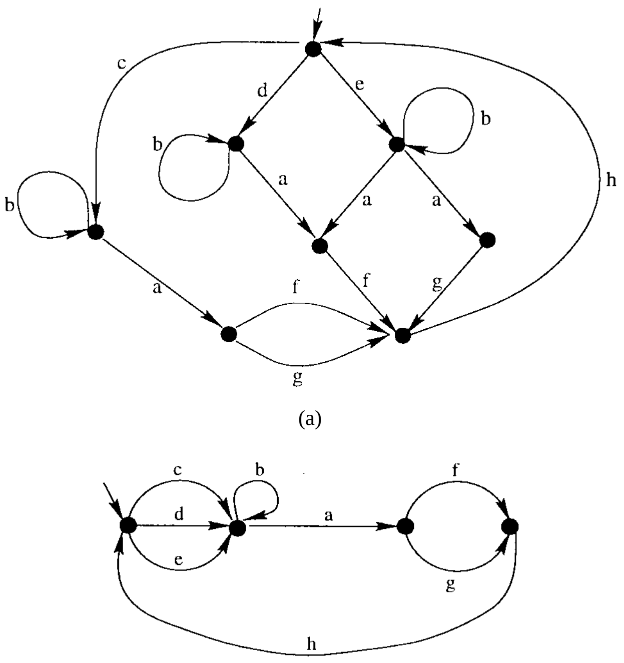
(b)

Fig. 2. A message transmission system.

the channel selection is nondeterministic, meaning that it is not under control of the sender.

The specification for legal behavior of the transmission system, given formally in Fig. 2(b), states that top-secret messages must be sent only on the secure channel, that secret messages can be sent controllably (that is, with control at the disposal of the sender) on either channel,[8] while no restriction is imposed on the channel selection for the transmission of nonsecret messages. More specifically, the specification states that if d has occurred, then following the occurrence of a, the event f (and only f) must be possible next. Similarly, if c has occurred, then following the occurrence of a, both f and g must be deterministically possible. In contrast, if e has occurred, then after a, either f or g can follow, but the choice is permitted to be nondeterministic. That is, there is no insistence that the channel selection be controllable.

It is noteworthy that the difference between the specification for secret and nonsecret messages is not a language difference. Indeed, the same event sequences are permitted in both cases. There is, however, a behavioral difference that is captured by the trajectory model as will be seen in detail later.



(a)

(b)

Fig. 3. Modified specification for message transmission system.

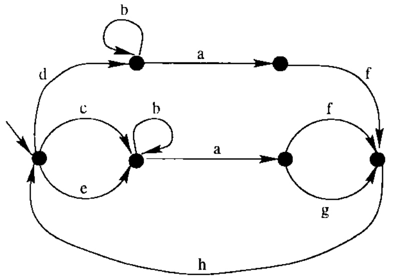


Fig. 4. $\hat{\mathcal{H}}_d$.

It is further noteworthy that for the statement of the specification (expressed by the automaton $\mathcal{H}$), there is no need to have a detailed model of the process. That is, $L(\mathcal{H})$ need not be, and in this example is not, a sublanguage of $L(\mathcal{P})$.

We proceed now with the analysis of our specification problem. First, we begin by constructing the modified specification automaton

$$\hat{\mathcal{H}} = \mathcal{H} \| \ \det(L(\mathcal{P}))$$

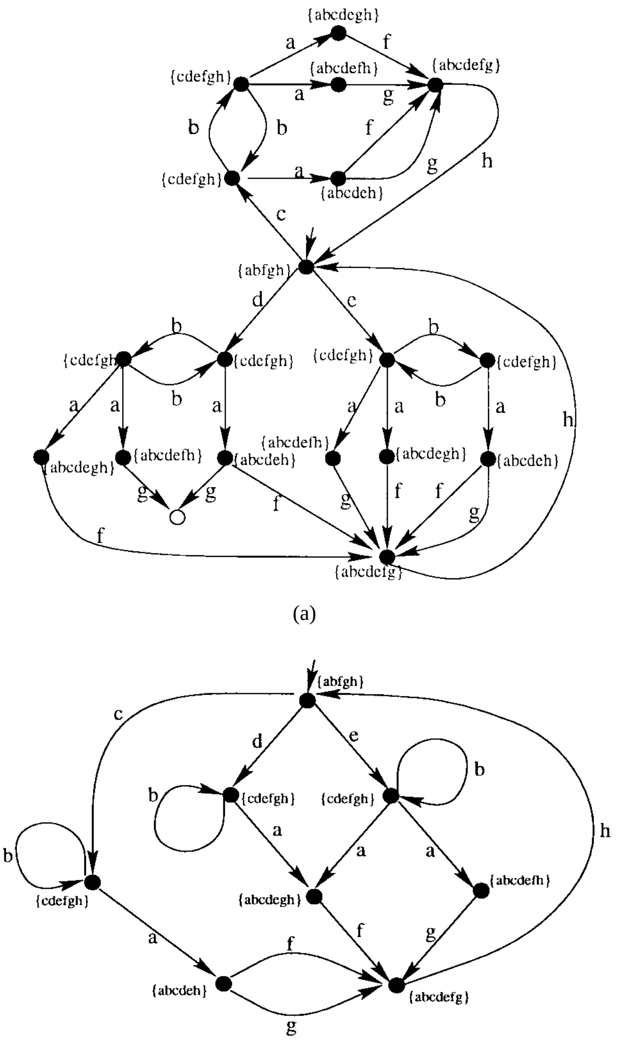
which together with $\det(L(\mathcal{P}))$ is shown in Fig. 3.

Next we construct the modified process $\overline{\mathcal{P}} = \mathcal{P} \| \overline{\mathcal{H}}_d$. The process $\hat{\mathcal{H}}_d$ is given in Fig. 4 and the process $\overline{\mathcal{P}}$ is shown in trajectory-model form, along with the trajectory-model representation of $\hat{\mathcal{H}}$, in Fig. 5.

The reader will note that the state marked by an unfilled circle in $\overline{\mathcal{P}}$ (which is reached by the string db^*ag) is an illegal state because it violates the language restriction imposed by the specification $\mathcal{H}$.

We shall see in the next section that if we use $\overline{\mathcal{P}}$ with this as the only illegal state, an optimal (minimally restrictive) supervisor can be obtained that guarantees satisfaction of the

[8] This allows for the possibility of implementing an additional supervisor to perform the channel selection.



(a)

(b)

Fig. 5. Trajectory models of plant and specification.

language restriction imposed by $\mathcal{H}$. To obtain the trajectory model specification as a static specification, we employ Algorithm 2 to $\overline{\mathcal{P}}$ and $\hat{\mathcal{H}}$. The resulting automaton is shown in Fig. 6.

In this automaton, two types of illegal states appear: states that violate the language constraints of the specification (marked by an unfilled circle) and states that violate the specification's trajectory-model constraints (marked by an encircled bullet).

The nondeterministic supervisory control problem with trajectory-model specification can now be stated in the following static framework: construct a supervisor (minimally restrictive, if possible) such that no illegal state is ever visited.

In the next section we pursue this example further to obtain the optimal supervisors and supervised plants. ▪

We conclude this section with a discussion and elaborations regarding some interesting special cases.

Deterministic Systems: First, let us consider the special case where the process $\mathcal{P}$ is deterministic. Suppose there are two specifications $\hat{\mathcal{H}}_1$ and $\hat{\mathcal{H}}_2$, such that $\hat{\mathcal{H}}_1 \subseteq \hat{\mathcal{H}}_2$ and $L = L(\hat{\mathcal{H}}_1) = L(\hat{\mathcal{H}}_2) \subseteq L(\mathcal{P})$. Since $\mathcal{R} = \det(L(\mathcal{R})) \subseteq \mathcal{H}$ for every trajectory model $\mathcal{H}$ satisfying $L(\mathcal{R}) = L(\mathcal{H})$, it follows that $\mathcal{P} \cap \hat{\mathcal{H}}_1 = \mathcal{P} \cap \hat{\mathcal{H}}_2$, so that for every path p in $\overline{\mathcal{P}}$, the associated trajectory t_p is in $\hat{\mathcal{H}}_1$ if and only if it is in $\hat{\mathcal{H}}_2$. This is true, in particular, if $\hat{\mathcal{H}}_2 = \mathrm{nondet}(L(\hat{\mathcal{H}}_1))$, which is simply the language specification L. Thus, in the case of deterministic systems, there is nothing to be gained by trajectory specifications beyond what can be specified and achieved by language specifications. The Ramadge–Wonham framework, which has been studied extensively in the literature, is complete and all that is needed in this case.

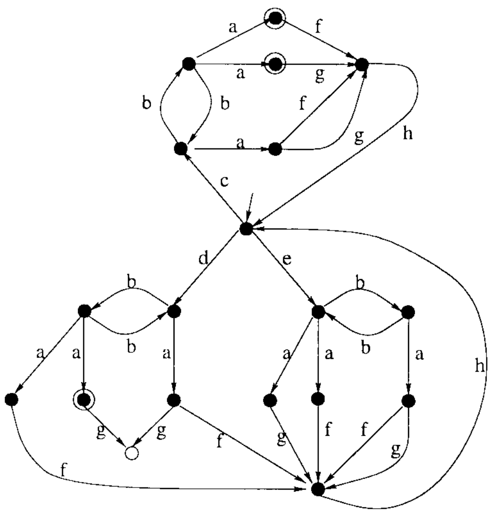


Fig. 6. Automaton $\overline{\mathcal{P}}$ with static specification.

Nondeterministic Systems with Language Specifications: We have already seen earlier that the language restriction imposed by a trajectory-model specification is embedded as a component in the translation process to the equivalent static specification framework and can be isolated as a separate supervisor synthesis problem (that satisfies only the language restriction). This will be demonstrated for our example in the next section. A further noteworthy observation is that if L is a language specification, then $\mathrm{nondet}(L)$ is the equivalent trajectory-model specification. That is, $\mathrm{nondet}(L)$ as a trajectory-model specification, yields precisely the same result as L as a language specification.

However, if we are only concerned with language specifications, we can proceed directly along a different and much simpler path.

We begin by lifting $\mathcal{P}$ to $\tilde{\mathcal{P}}$ and then letting

$$E = L(\tilde{\mathcal{P}}) \cap P^{-1}L(\mathcal{H}).$$

The supervisor synthesizing E will satisfy the language specification. To show this, let us use the following.

Lemma 2: Given two languages $A \subseteq \Sigma^*$ and $B \subseteq (\Sigma \cup \Sigma')^*$. If $A \subseteq PB$, then

1) $P(B \cap P^{-1}A) = A$;
2) $B \cap P^{-1}A$ is normal with respect to B.

Proof: The proof is elementary. ■

Now we can prove the following.

Theorem 5: If the supervisor γ synthesizes E, that is, if

$$L(\gamma/\tilde{\mathcal{P}}) = E$$

then

$$L(\gamma/\mathcal{P}) = L(\mathcal{H}).$$

Proof:

$$\begin{aligned} L(\gamma/\mathcal{P}) &= PL(\gamma/\tilde{\mathcal{P}}) \\ &= PE \\ &= P(L(\tilde{\mathcal{P}}) \cap P^{-1}L(\mathcal{H})) \\ &= L(\mathcal{H}). \end{aligned}$$

The last equality is the consequence of Lemma 2 and the fact that $L(\mathcal{H}) \subseteq L(\mathcal{P}) = PL(\tilde{\mathcal{P}})$. ■

As shown in [18], since E is normal, E is controllable with respect to $L(\tilde{\mathcal{P}})$ if and only if PE is controllable with respect to $PL(\tilde{\mathcal{P}}) = L(\mathcal{P})$. Therefore, we have the following.

Corollary 1: For a nondeterministic system $\mathcal{P}$ and a language specification $L(\mathcal{H})$, there exists a supervisor γ such that $L(\gamma/\mathcal{P}) = L(\mathcal{H})$, if and only if $L(\mathcal{H})$ is controllable with respect to $L(\mathcal{P})$. ■

This corollary leads naturally to the results of [26], where only language specifications are considered.[9]

VI. Supervisor Synthesis

In Section IV we have shown how a supervisor can be synthesized for a nondeterministic system with static specifications using the lifting procedure Extend, in case the legal language E is controllable and observable. In Section V we have shown that the supervisory control problem for a nondeterministic system with dynamic specifications can be translated to an equivalent problem with static specifications.

When the controllability and observability conditions are not satisfied for the language E, then no supervisor exists that achieves the exact specification. In the present section we focus our attention on obtaining the best approximation of the optimal supervisor; that is, we shall show how we can synthesize the minimally restrictive supervisor that confines the supervised systems to its subset of good states.

We shall assume that the problem is already formulated as one with static specifications. That is, we assume that the system and specification are described by

$$\overline{\mathcal{P}} = (\Sigma \cup \{\epsilon\}, Q, \delta, q_o, Q_b)$$

where Q_b is the set of bad states that must be avoided.[10]

First we lift $\overline{\mathcal{P}}$ to $\tilde{\mathcal{P}}$ using the procedure Extend and define E as in Section IV. If E is not controllable and observable with respect to $L(\tilde{\mathcal{P}})$, then we will find the largest sublanguage of E that is controllable and observable and synthesize a supervisor based on that language.

As we will show, this largest sublanguage always exists and is the supremal controllable and normal sublanguage of E. By synthesizing a supervisor based on this sublanguage, we allow $\tilde{\mathcal{P}}$ to visit as many good states as possible without violating the specification. The supervised system obtained this way is described by the largest possible legal subautomaton of $\tilde{\mathcal{P}}$. Since a larger subautomaton of $\tilde{\mathcal{P}}$ projects to a larger subautomaton of $\overline{\mathcal{P}}$, the supervisor thus synthesized generates the largest possible legal subautomaton of $\overline{\mathcal{P}}$, and the corresponding supervisor is thus minimally restrictive.[11]

Our first design procedure is given by the following.

Algorithm 3 *(Off-Line Synthesis):*

1) Extend($\overline{\mathcal{P}} \rightarrow \tilde{\mathcal{P}}$).
2) Sup$\mathcal{CN}(\tilde{\mathcal{P}} \rightarrow \tilde{\mathcal{P}}_{nc})$.
3) Design a supervisor off-line.

In Algorithm 3, Step 1) lifts $\overline{\mathcal{P}}$ to $\tilde{\mathcal{P}}$ as described earlier. Step 2) calculates the supremal controllable and normal sublanguage of E. A formula for calculating the supremal controllable and normal sublanguage $\sup \mathcal{CN}(E)$ is given in [1]. We note that since in $\tilde{\mathcal{P}}$, all controllable events are observable ($\Sigma_c \subseteq \Sigma = \Sigma_o$), controllability and observability of a language is equivalent to controllability and normality of that language. Step 3) designs a supervisor off-line in the usual way [17]. Therefore, we can easily prove the following.

Proposition 9: Let γ_1 be the supervisor synthesized by Algorithm 3. Then

$$L(\gamma_1/\tilde{\mathcal{P}}) = \sup \mathcal{CN}(E).$$

■

There is another approach that can be used for supervisor synthesis, that is to design a supervisor on-line instead of off-line. The advantage of the on-line approach is that the computational complexity is linear at each step of event execution [8].

Algorithm 4 *(On-Line Synthesis):*

1) Extend($\overline{\mathcal{P}} \rightarrow \tilde{\mathcal{P}}$).
2) Sup$\mathcal{C}(\tilde{\mathcal{P}} \rightarrow \tilde{\mathcal{P}}_c)$.
3) Design a supervisor on-line.

Step 1) is same as that of Algorithm 3. Step 2 calculates the supremal controllable sublanguage of the legal language E. This can be done with linear complexity for a closed language E. All that needs to be done is to successively delete the states from which $\tilde{Q}_b$ can be reached via strings of uncontrollable transitions

$$\tilde{\mathcal{P}}_c = (\Sigma \cup \Sigma', \tilde{Q}_c, \tilde{\delta}|_{\tilde{Q}_c}, q_o)$$

where

$$\tilde{Q}_c = \{\tilde{q} \in \tilde{Q}\colon (\forall\, u \in \Sigma_{uc}^*)\tilde{\delta}(\tilde{q}, u) \notin \tilde{Q}_b\}.$$

In Step 3), we design a supervisor on-line using the results of [8]. The resulting supervisor will generate the supremal controllable and normal sublanguage of E and hence allows the system to visit as many legal states as possible. As shown in [8], the complexity at each step of event execution is linear in $|\tilde{Q}|$.

The correctness of the above algorithm can be easily proved and is summarized in the following.

[9] In [26], the system model has been allowed to include also "driven" as well as unobservable events. Thus, our corollary relates only to their case where driven and unobservable events are absent. The case where unobservable events are present in the model is discussed for general trajectory-model specifications in [9].

[10] The reader will note that in Section VI the subset $\overline{Q}_t$ of marked states consisted of the good states so that $\overline{Q}_b = \overline{Q} - \overline{Q}_t$.

[11] Note that the trajectory model of the supervised system thus synthesized is not, in general, a subset of $\mathcal{P}_s$, the trajectory model of the specification. This is because, in general, the trajectory model of a subautomaton is not a subset of the trajectory model of the larger automaton.

Proposition 10: Let γ_2 be the supervisor synthesized by Algorithm 4. Then

$$L(\gamma_2/\tilde{\mathcal{P}}) = \sup \mathcal{CN}(E).$$

The supervisors thus obtained are "optimal" in the sense that they are minimally restrictive, as stated below.

Theorem 6: The supervisors designed using Algorithms 3 and 4 are minimally restrictive and allow the supervised system to visit as many legal states as possible.

Proof: By Propositions 9 and 10, the supervisors generate the supremal controllable and normal sublanguage of E, $\sup \mathcal{CN}(E)$. Since all the unobservable events Σ' are artificial and hence uncontrollable, by Proposition 1, controllability and normality is equivalent to controllability and observability. Therefore, the supremal controllable and normal sublanguage of E is also the supremal controllable and observable sublanguage of E. Hence the supervisors generate the largest legal sublanguage of E and are minimally restrictive. ■

By the above theorem, the supervised (nondeterministic) systems $\gamma_1/\overline{\mathcal{P}}$ and $\gamma_2/\overline{\mathcal{P}}$ are the same and are described by the largest possible subautomaton of $\overline{\mathcal{P}}$.

In view of the way $\sup \mathcal{CN}$ is calculated [1], we can modify Algorithm 3 by directly converting $\mathcal{P}$ to a deterministic automaton $\hat{\mathcal{P}}$ without adding the unobservable events Σ'. This leads to the following direct approach.

Algorithm 5 *(Direct Synthesis):*

1) Convert($\overline{\mathcal{P}} \rightarrow \hat{\mathcal{P}}$).
2) Sup$\mathcal{C}(\hat{\mathcal{P}} \rightarrow \hat{\mathcal{P}}_c)$.
3) Design a supervisor off-line.

■

Procedure Convert, that converts the nondeterministic automaton $\overline{\mathcal{P}}$ into a deterministic one $\hat{\mathcal{P}}$, is standard [12]. Each state in $\hat{\mathcal{P}}$ is now a subset of states in $\overline{\mathcal{P}}$. We call such a state "bad" if it includes a bad state of $\overline{\mathcal{P}}$

$$\hat{\mathcal{P}} = Acc(\Sigma, \hat{Q}, \hat{\delta}, \hat{q}_o, \hat{Q}_b)$$

where

$$\begin{aligned}\hat{Q} &= 2^Q\\ \hat{\delta}(\hat{q}, \sigma) &= \{q' \in Q\colon (\exists q \in \hat{q}) q' \in \epsilon^*(\delta(q, \sigma))\}\\ \hat{q}_o &= \{q' \in Q\colon q' \in \epsilon^*(q_o)\}\\ \hat{Q}_b &= \{\hat{q} \in \hat{Q}\colon \hat{q} \cap Q_b \neq \emptyset\}.\end{aligned}$$

Step 2) computes the supremal controllable sublanguage

$$\hat{\mathcal{P}}_c = (\Sigma, \hat{Q}_c, \hat{\delta}|_{\hat{Q}_c}, \hat{q}_o)$$

where

$$\hat{Q}_c = \{\hat{q} \in \hat{Q}\colon (\forall u \in \Sigma_{uc}^*) \hat{\delta}(\hat{q}, u) \notin \hat{Q}_b\}.$$

Step 3) then designs a supervisor based on $\hat{\mathcal{P}}_c$.

We will show that the resulting supervised systems using Algorithms 3 and 5 are the same as far as strings in Σ^* are concerned.

Theorem 7: The supervisor designed using Algorithm 5 is minimally restrictive and allows the supervised system to visit as many legal states as possible.

Proof: Let γ_1 and γ_3 be the supervisors obtained from Algorithms 3 and 5, respectively. In view of Theorem 6, it suffices to prove that

$$\gamma_1/\overline{\mathcal{P}} = \gamma_3/\overline{\mathcal{P}}.$$

By Proposition 9

$$L(\gamma_1/\overline{\mathcal{P}}) = P \sup \mathcal{CN}(E).$$

By a formula in [1]

$$\sup \mathcal{CN}(E) = L(\tilde{\mathcal{P}}) \cap P^{-1} \sup \mathcal{C}_L(P \sup \mathcal{N}(E))$$

where $\mathcal{C}_L$ denotes controllability with respect to $L(\overline{\mathcal{P}})$. Therefore, by Lemma 2

$$L(\gamma_1/\overline{\mathcal{P}}) = \sup \mathcal{C}_L(P \sup \mathcal{N}(E)).$$

Again, by another formula in [1]

$$\sup \mathcal{N}(E) = E - P^{-1}P(L(\tilde{\mathcal{P}}) - E)(\Sigma \cup \Sigma')^*.$$

Now, it is clear that Step 1) of Algorithm 5 computes $P \sup \mathcal{N}(E)$, and Step 2) of Algorithm 5 computes $\sup \mathcal{C}_L$ $(P \sup \mathcal{N}(E))$. Hence

$$L(\gamma_1/\overline{\mathcal{P}}) = L(\gamma_3/\overline{\mathcal{P}})$$

which implies

$$\gamma_1/\overline{\mathcal{P}} = \gamma_3/\overline{\mathcal{P}}.$$

■

Remark: The theory of the present paper was developed under the assumption that the system under consideration is nondivergent. However, the nondivergence assumption is not essential for our theory and was made here primarily to render the paper more accessible to the reader, since the analysis of divergent systems is much more complicated. The reader can consult [7] for details regarding trajectory models of systems with divergence. Finally, although the assumption of nondivergence is very reasonable in most practical cases, the algorithmic framework presented in the present paper is valid for systems with divergence (i.e., automata with ϵ-cycles) as well. ■

We conclude this section with a continuation of Example 1.

Example 1—Continued: In Fig. 6 we have given the automaton $\overline{\mathcal{P}}$ with static specifications depicting both the constraints imposed by the language of $\mathcal{H}$ (the only illegal state being the unfilled circle) and the constraints imposed by $\mathcal{H}$ as a trajectory specification (the illegal states being the ones marked by unfilled circles or encircled bullets).

When applying the procedure Extend to the automaton $\overline{\mathcal{P}}$, we obtain the automaton $\tilde{\mathcal{P}}$ shown in Fig. 7.

When any of the above synthesis methods is used with respect to the language specification of $\tilde{\mathcal{P}}$ (that is, the state marked by unfilled circles being the only illegal state), the supervisor γ is obtained as

$$\gamma(s) = \begin{cases} \{g\}, & \text{if } s = db^*a\\ \emptyset, & \text{otherwise.}\end{cases}$$

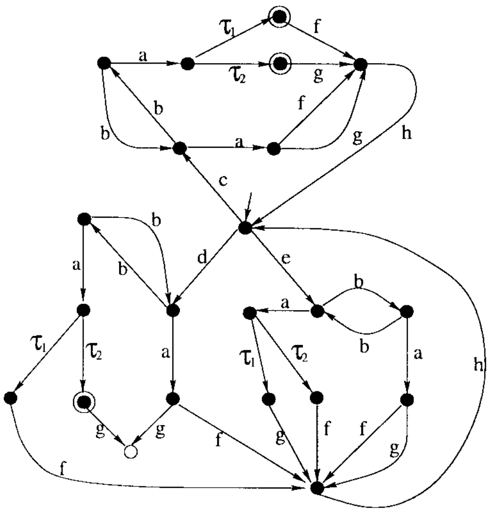


Fig. 7. Lifted $\tilde{\mathcal{P}}$ with static specification.

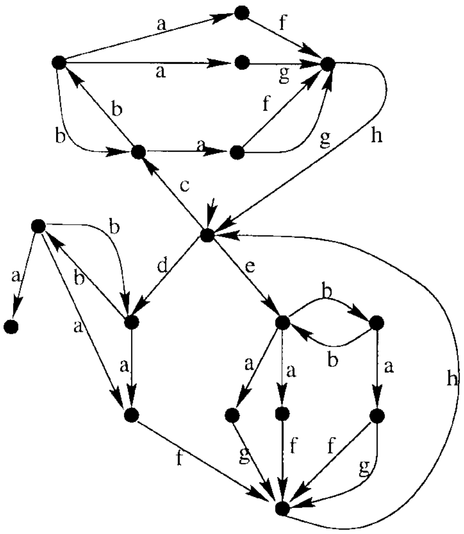


Fig. 8. Controlled system for language specification.

Here and below we assume that all events a, b, c, d, e, f, g, h are controllable. The controlled system will then be obtained as the automaton shown in Fig. 8.

For the trajectory specification of $\tilde{\mathcal{P}}$ the supervisor is obtained as

$$\gamma(s) = \begin{cases} \{a\}, & \text{if } s = cb(bb)^* \\ \{a\}, & \text{if } s = db(bb)^* \\ \{g\}, & \text{if } s = d(bb)^*a \\ \emptyset, & \text{otherwise} \end{cases}$$

and the controlled system is obtained as in Fig. 9.

The only restriction that the language specification imposes is the prevention of the event g from occurring after a string that includes a recent d. That is, it prevents top-secret messages from being sent via the nonsecure channel. The reader will note, however, that the (language) specification as discussed here does not concern itself with the issue of deadlock. Therefore, it permits the possible occurrence of deadlock after $d(bb)^*ba$.[12]

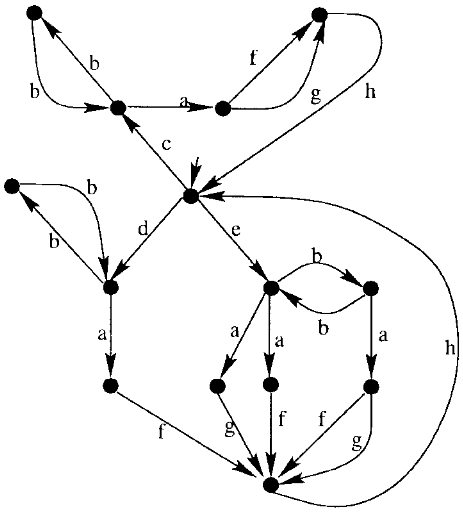


Fig. 9. Controlled system for trajectory specification.

The reader will note that with the trajectory specification, the supervisor distinguishes between rather subtle differences in requirements for the three types of messages. Specifically, deadlock no longer occurs following the transmission of top-secret messages because the supervisor disables transmission of these messages altogether from terminal 3 (disablement of a following $db(bb)^*$). The supervisor also distinguishes between secret and nonsecret messages. It disables transmission of secret messages from terminal 3 while imposing no restrictions on transmission of nonsecret messages. It should also be noted that the restriction imposed on secret messages does not curtail the generated language. Rather, it restricts the degree of permitted nondeterminism in the controlled system. ■

VII. Conclusion

We studied the supervisory control for nondeterministic DES's subject to both language and trajectory-model specifications. We have shown that there is a close relation between the problem of control of a nondeterministic system and the problem of control under partial observation of related deterministic system that can be derived from the original process and specification. Thus, our approach was to translate the given supervisory control problem into an equivalent problem for partial observation systems. In view of this relation, we developed a uniform theory for both deterministic and nondeterministic systems that enables the application of known results regarding control of partial observation systems to the control of nondeterministic systems as well. This is true especially with respect to supervisor synthesis methods that are difficult to develop directly in the nondeterministic setting. However, before this translation can be carried out, the subtle differences between language specifications and trajectory specifications (unique to nondeterministic systems) had to be handled carefully. We demonstrated the subtleties of

[12] Nonblocking supervisory control of nondeterministic systems is investigated in detail in [9].

trajectory-model specifications by an example and developed an algorithm for incorporating the trajectory-model specification. Finally, while our theory was developed via a lifting procedure, we have shown that the actual synthesis of a supervisor can be carried out with or without lifting.

References

[1] R. D. Brandt, V. Garg, R. Kumar, F. Lin, S. I. Marcus, and W. M. Wonham, "Formulas for calculating supremal controllable and normal sublanguages," *Syst. Contr. Lett.*, vol. 15, pp. 111–117, 1990.

[2] S. L. Chung, S. Lafortune, and F. Lin, "Limited lookahead policies in supervisory control of discrete event systems," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 1921–1935, Dec. 1992.

[3] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya, "Supervisory control of discrete-event processes with partial observations," *IEEE Trans. Automat. Contr.*, vol. 33, pp. 249–260, Mar. 1988.

[4] M. Fabian and B. Lennartson, "Object oriented supervisory control with a class of nondeterministic specifications," Chalmers Univ. Technol., Goteborg, Sweden, Rep. CTH/RT/I-94/007, 1994.

[5] ——, "On nondeterministic supervisory control," in *Proc. 35th Conf. Decision Contr.*, Kobe, Japan, 1996, pp. 2213–2218.

[6] M. Heymann, "Concurrency and discrete-event control," *IEEE Contr. Syst. Mag.*, vol. 10, no. 4, pp. 103–112, 1990.

[7] M. Heymann and G. Meyer, "An algebra of discrete-event processes," NASA Technical Memo. 102848, 1991.

[8] M. Heymann and F. Lin, "On-line control of partially observed discrete event systems," *Discrete Event Dynamic Syst.: Theory Appl.*, vol. 4, no. 3, pp. 221–236, 1994.

[9] ——, "Nonblocking supervisory control of nondeterministic systems," Technion CIS Rep. 9620, Oct. 1996.

[10] ——, "On observability and nondeterminism in discrete event control," in *Proc. 33rd Allerton Conf. Communication, Contr., and Computing*, 1995, pp. 136–145.

[11] ——, "Discrete event control of nondeterministic discrete event systems," in *Proc. 35th IEEE Conf. Decision Contr.*, 1996, pp. 4445–4450.

[12] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. New York: Addison-Wesley, 1979.

[13] K. Inan, "Nondeterministic supervision under partial observation," in *11th International Conference on Analysis and Optimization of Systems*, G. Cohen and J.-P. Quadrat, Eds. New York: Springer-Verlag, 1994, pp. 39–48.

[14] R. Kumar and M. A. Shayman, "Non-blocking supervisory control of nondeterministic systems via prioritized synchronization," *IEEE Trans. Automat. Contr.*, vol. 41, pp. 1160–1175, Aug. 1996.

[15] ——, "Supervisory control of nondeterministic systems under partial observation and decentralization," *SIAM J. Contr. Optimization*, to be published.

[16] F. Lin, "On controllability and observability of discrete event systems," Ph.D. dissertation, Dept. Electrical Engineering, Univ. Toronto, 1987.

[17] F. Lin and W. M. Wonham, "On observability of discrete event systems," *Inform. Sci.*, vol. 44, no. 3, pp. 173–198, 1988.

[18] ——, "Decentralized supervisory control of discrete-event systems," *Inform. Sci.*, vol. 44, no. 3, pp. 199–224, 1988.

[19] ——, "Decentralized control and coordination of discrete event systems with partial observation," *IEEE Trans. Automat. Contr.*, vol. 35, pp. 1330–1337, Dec. 1990.

[20] ——, "Supervisory control of timed discrete event systems under partial observation," *IEEE Trans. Automat. Contr.*, vol. 40, pp. 558–562, Mar. 1994.

[21] A. Overkamp, "Supervisory control for nondeterministic systems," in *Proc. 11th Int. Conf. Analysis Optimiz. Syst.*, 1994, pp. 59–65.

[22] ——, "Supervisory control using failure semantics and partial specification," *IEEE Trans. Automat. Contr.*, vol. 42, pp. 498–510, Apr. 1997.

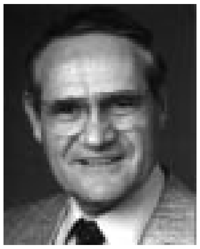
[23] R. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Contr. Optim.*, vol. 25, no. 1, pp. 206–230, 1987.

[24] ——, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[25] K. Rudie and W. M. Wonham, "Think globally, act locally: Decentralized supervisory control," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 1692–1708, Nov. 1992.

[26] M. Shayman and R. Kumar, "Supervisory control of nondeterministic systems with driven events via prioritized synchronization and trajectory models," *SIAM J. Contr. Optim.*, vol. 33, no. 2, pp. 469–497, 1995.

[27] J. N. Tsitsiklis, "On the control of discrete-event dynamical systems," *Math. Contr., Signals, Syst.*, vol. 2, no. 1, pp. 95–107, 1989.



Michael Heymann received the B.Sc. and M.Sc. degrees from the Technion-Isreal Institute of Technology, Haifa, Israel, in 1960 and 1962, respectively, as well as the Ph.D. degree from the University of Oklahoma, Norman, in 1965, all in chemical engineering.

During 1965–1966, he was on the faculty of the University of Oklahoma. From 1966 to 1968 he was with Mobil Research and Development Corporation engaged in research on control and systems theory. From 1968 to 1970 he was with the Ben Gurion University of the Negev, Beer Sheva, where he established and headed the Chemical Engineering Department. Since 1970 he has been with the Technion, where he is a Professor in the Department of Computer Science, holding the Carl Fechheimer Chair. He has previously been with the Department of Electrical Engineering and Chairman of the Department of Applied Mathematics. He held visiting positions at the University of Toronto, the University of Florida, the University of Eindhoven, Concordia University, CSIR, Yale University, the University of Bremen, and the University of Newcastle. Since 1983 he has been associated with NASA Ames Research Center where he spent three Sabbatical leaves and many summers as an NRC-Senior Research Associate. His current research interests include discrete-event systems, hybrid systems, the theory of concurrent processes, and problems related to interactive automation.

Dr. Heymann is on the editorial board of the *SIAM Journal on Control and Optimization*.

Feng Lin (S'86–M'87) received the B.Eng. degree in electrical engineering from Shanghai Jiao-Tong University, Shanghai, China, in 1982, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Ontario, Canada, in 1984 and 1988, respectively.

From 1987 to 1988, he was a Postdoctoral Fellow at Harvard University, Cambridge, MA. Since 1988, he has been with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI, where he is currently an Associate Professor. He also worked at NASA Ames Research Center in the summers of 1992 and 1996. His research interests include discrete-event systems, hybrid systems, robust control, image processing, and neural networks.

Dr. Lin co-authored a paper that received a George Axelby Outstanding Paper Award from IEEE Control Systems Society. He is also the recipient of a Research Initiation Award from the National Science Foundation, an Outstanding Teaching Award from Wayne State University, a Faculty Research Award from ANR Pipeline Company, and a Research Award from Ford Motor Company. He is an Associate Editor of IEEE Transactions on Automatic Control.