Proceedings of the 35th
Conference on Decision and Control
Kobe, Japan • December 1996

Discrete Event Control of Nondeterministic Systems

Michael Heymann[1] and Feng Lin[2]

Abstract

Nondeterminism in discrete-event systems may occur as a result of partial modeling. For the adequate description of nondeterministic systems and nondeterministic phenomena, the trajectory-model formalism that was introduced in [2] [4] is employed. In the present paper we develop a theory of supervisory control for nondeterministic discrete-event systems subject to trajectory-model specifications. We show how well known algorithms for supervisory control (of deterministic systems) under partial observation can be adapted for synthesis of supervisors for nondeterministic systems.

1 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 [14]. A great deal of attention was also given to the control of partially observed discrete-event systems [10], 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, algorithms for supervisor synthesis, for off-line as well as on-line implementation, 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 discrete-event systems, 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, also the degree of nondeterminism that the controlled system is allowed to retain. The *trajectory model* formalism was introduced in [2] and [4] as a semantic framework for modeling and specification of nondeterministic behaviors, and it was shown to adequately capture nondeterministic behaviors that one might wish to discriminate and distinguish by discrete-event control. Thus, for control purposes, nondeterministic discrete-event systems can be modeled either as nondeterministic automata (with ϵ-transitions) or as *trajectory models*.

In recent years, there has been increasing interest in supervisory control of nondeterministic systems as reported, e.g., in [12] [13] [15] [9] [1]. However, while some existence conditions for control of nondeterministic systems have been derived, few explicit algorithms for supervisor synthesis have been reported. Indeed, the direct supervisor synthesis for nondeterministic systems seems to be quite a difficult task.

Motivated by this observation, we began an investigation, [7] [5] [6], of the connection between the supervisory control problem for general nondeterministic systems and the corresponding problem for partially observed deterministic systems. Our work led us to develop an approach to synthesis of supervisors for nondeterministic systems wherein direct advantage is taken of the existing theory for control under partial observation.

In the present paper we consider the supervisory control problem of nondeterministic discrete-event systems subject to trajectory-model specifications. 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 spec-

[1]Department of Computer Science, Technion, Israel Institute of Technology, Haifa 32000, Israel. Work completed while the author was a Senior NRC Research Associate at NASA Ames Research Center, Moffett Field, CA 94035 and supported in part by the Technion Fund for Promotion of Research.

[2]Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202. Work supported in part by the National Science Foundation under grant ECS-9315344.

0-7803-3590-2/96 $5.00 © 1996 IEEE

ifications. 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 [5] and we only 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 [5], 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, 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 to be addressed extensively in [7].

In Section 2 we briefly review the relevant aspects of the theory of supervisory control under partial observation, in Section 3 we review the main concepts of nondeterministic discrete-event systems 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 4 we review the supervisory control of nondeterministic systems with static (state-based) specifications, in which the specification of legal behavior is given as a subset of legal states. In Section 5 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 6 we conclude the paper with a brief discussion of the methodology for supervisor synthesis.

2 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 a (deterministic) automaton $G = (\Sigma, Q, \delta, q_o)$ with elements defined in a usual way. The language generated by G is denoted by $L(G)$. The event set is partitioned into controllable (observable) and uncontrollable (unobservable) disjoint subsets: $\Sigma = \Sigma_c \cup \Sigma_{uc}$ $(= \Sigma_o \cup \Sigma_{uo})$. A supervisor is a disablement map $\gamma : PL(G) \to 2^{\Sigma_c}$ (where $P : \Sigma^* \to \Sigma_o^*$ 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 $\sigma \in \Sigma$ that are disabled by the supervisor. The language generated by the supervised system is denoted by $L(\gamma/G)$.

We say that a language K is closed, if it equals its prefix closure. We also define controllability, normality and observability as in [14] [10]. The basic supervisor existence result is that for a nonempty language $K \subseteq L(G)$ there exists a supervisor γ such that $L(\gamma/G) = K$ if and only if K is closed, controllable and observable. 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 [11].

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

An important property of controllable (normal) languages is that the supremal controllable and normal sublanguage $sup\mathcal{CN}(E)$ of a language E exists.

3 Nondeterminism

In this section we briefly review the trajectory-model formalism of [4] (see also [15]).

A *trajectory* is a record associated with a run of a system G. It is more detailed than a trace $s \in L(G)$ in that it lists, in addition to the successfully executed events, also 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

$$t = (X_0, \sigma_1, X_1, ..., X_{k-1}, \sigma_k, X_k),$$

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. The trace associated with t is defined as $tr(t) = \sigma_1 ... \sigma_k$.

A trajectory is called *valid* if $\sigma_i \notin X_{i-1}$ for all $i > 0$. For a trajectory $t = (X_0, (\sigma_1, X_1), ..., (\sigma_k, X_k))$, a trajectory r is a *prefix* of t, denoted $r \preceq t$, if

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

and $0 \leq j \leq k$. The set of all prefixes of t is called the *prefix-closure* of t and is denoted $pref(t)$. A trajectory r is said to be *dominated* by t, denoted $r \sqsubseteq t$, if it is of the form $r = (Y_0, (\mu_1, Y_1), ..., (\mu_k, Y_k))$, with $\mu_i = \sigma_i$ for $1 \leq i \leq k$, and $Y_j \subseteq X_j$ for $0 \leq j \leq k$. The set of all

trajectories dominated by t is called the *completion*, or *dominance-closure*, of t and denoted $comp(t)$. Finally, we define the *closure* of t, $cl(t)$, as

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

and the closure of a set of trajectories $\mathcal{T}$ as $cl(\mathcal{T}) := \bigcup_{t \in \mathcal{T}} cl(t)$. We say that a set of trajectories $\mathcal{T}$ is *closed*[3] if $\mathcal{T} = cl(\mathcal{T})$. We say that a set of trajectories $\mathcal{T}$ is *saturated* if the following condition holds:

$$\begin{array}{l}(\forall k = 1, 2, ...)(\forall j : 0 \leq j \leq k)(\forall \sigma \in \Sigma - X_j) \\ (((X_0, (\sigma_1, X_1), ..., (\sigma_k, X_k)) \in \mathcal{T} \\ \wedge (X_0, (\sigma_1, X_1), ..., (\sigma_j, X_j)(\sigma, \emptyset)) \notin \mathcal{T}) \\ \Rightarrow (X_0, (\sigma_1, X_1), ..., (\sigma_j, X_j \cup \{\sigma\}) ... (\sigma_k, X_k)) \in \mathcal{T}).\end{array}$$

Definition 1 [4] A (possibly) nondeterministic *process* $\mathcal{P}$ is a closed and saturated subset $\mathcal{P} \subseteq 2^{\Sigma} \times (\Sigma \times 2^{\Sigma})^*$.

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 dominant trajectories in $\mathcal{T}$ is called the *dominance-set* of $\mathcal{T}$ and is denoted $dom(\mathcal{T})$. The following proposition states that a process $\mathcal{P}$ is completely characterized by its dominance set.

Proposition 2 Let $\mathcal{P}$ be a process. Then $cl(dom(\mathcal{P})) = \mathcal{P}$.

We shall next examine how trajectory-model representations of discrete event systems are related to their representation as automata. Consider a discrete-event system 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 transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$. Assume, further, that the system is nondivergent. We proceed now to obtain the set of trajectories associated with $\mathcal{P}$. First, we associate with each state $q \in Q$ its *maximal-refusal-set* $X_q \subseteq \Sigma$,

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

where $\epsilon^*(q)$, the ϵ−*closure* of q, is defined inductively [8] as

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

With each path $p = (q_0, \sigma_1, q_1, ..., \sigma_k, q_k)$ 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 $\tilde{t}_p := (X_{q_0}, \sigma_1, X_{q_1}, ..., \sigma_k, X_{q_k})$. (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. Denoting the set of trajectories t_p associated with all ϵ−stable paths in $\mathcal{P}$ by $dom(\mathcal{P})$ (a path $p = (q_0, \sigma_1, q_1, ..., \sigma_k, q_k)$ is *ϵ−stable* if no ϵ-transitions emanate from q_k), the trajectory model of $\mathcal{P}$ (which we also denote $\mathcal{P}$) is obtained as $\mathcal{P} = cl(dom(\mathcal{P}))$.

Conversely, we shall recall [4] that we can construct a nondeterministic state machine (represented as a transition graph with ϵ-transitions) directly from the set $dom(\mathcal{P})$ or, more specifically, from the set $\mathcal{M}(\mathcal{P})$ defined as

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

In view of the correspondence between trajectory models and nondeterministic automata, we can use either of them to model a nondeterministic system. We shall use the same symbol to denote both the nondeterministic automaton and its associated trajectory model. The languages generated and marked by a nondeterministic automaton $\mathcal{R}$ are denoted by $L(\mathcal{R})$ and $L_m(\mathcal{R})$ respectively.

Let $L \subseteq \Sigma^*$ be a prefix-closed language and let $\mathcal{TM}(L)$ denote the set of all trajectory models that share L as their trace set. Then if $\mathcal{P}$ and $\mathcal{Q}$ are two processes in $\mathcal{TM}(L)$, we are justified in saying that $\mathcal{P}$ is *more nondeterministic* than $\mathcal{Q}$ whenever $\mathcal{Q} \subseteq \mathcal{P}$ [4].

Definition 2 A process $\mathcal{P}$ is called *deterministic* if for every trajectory $(X_0, (\sigma_1, X_1), ..., (\sigma_k, X_k)) \in \mathcal{P}$ and any $\sigma \in \Sigma$

$$\begin{array}{l}(X_0, (\sigma_1, X_1), ..., (\sigma_k, X_k), (\sigma, \emptyset)) \in \mathcal{P} \Leftrightarrow \\ (X_0, (\sigma_1, X_1), ..., (\sigma_k, X_k \cup \{\sigma\})) \notin \mathcal{P}.\end{array}$$

The (unique) smallest process in $\mathcal{TM}(L)$ is deterministic [4] [6] and denoted $det(L)$, and the largest process in $\mathcal{TM}(L)$ (the union of all such processes) is denoted $nondet(L)$. An algorithm for construction of $det(L)$ is given in [4].

Consider a nondeterministic automaton, possibly with ϵ−transitions,

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

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'. This "lifting" 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 $\Sigma' = \{\tau_1, \tau_2, ...\}$.

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

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

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 ϵ–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 ϵ–transitions by transitions labeled $\tau_1, \tau_2, ...$ as follows:
   If $\tilde{\delta}(q,\epsilon) = \{q_1, ..., q_n\}$, then set
   $$\tilde{\delta}(q,\tau_1) := \{q_1\};$$
   ...
   $$\tilde{\delta}(q,\tau_n) := \{q_n\};$$
4. End of Procedure

4 Supervisory control with static specifications

In the present section we briefly review the supervisory control problem of nondeterministic systems subject to *static* (i.e. state-based) specifications [5]. Specifically, suppose that the system under consideration is modeled as a nondeterministic automaton

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

where $Q_b \subseteq Q$, $q_0 \notin Q_b$, is a subset of *forbidden* states that the system is not allowed to visit. Control is achieved by a supervisor $\gamma : L(\mathcal{P}) \to 2^{\Sigma_c}$, where $\gamma(s)$ 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 *legal* supervisor γ such that the supervised system $\gamma/\mathcal{P}$ never visits a state in Q_b. The supervisor γ is also required to be least restrictive in the sense that for every other legal supervisor γ' and every $s \in L(\mathcal{P})$, $\gamma(s) \subseteq \gamma'(s)$.

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

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

Then the trajectory model $\gamma/\mathcal{P}$ of the supervised system is obtained as

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

where $.||.$ denotes the strict synchronous composition of trajectory models [4] [6]. Next, we *lift* $\mathcal{P}$ 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

$$\tilde{Q}_b = Q_b \cup \{q \in \tilde{Q} - Q : \tilde{\delta}(q,\epsilon) \subseteq Q_b\}.$$

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, $E = \{s \in L(\tilde{\mathcal{P}}) : (\forall t \leq s)\tilde{\delta}(q_o, t) \notin \tilde{O}_b\}$. It then follows that $\tilde{\mathcal{P}}\backslash_{\Sigma'} = \mathcal{P}$.

Furthermore, the supervisor that restricts $\tilde{\mathcal{P}}$ to the supremal controllable and normal sublanguage of E is also the least restrictive (optimal) legal supervisor for $\mathcal{P}$. Thus, 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.

5 Supervisory control with dynamic specifications

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

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

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

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

We shall next explain in some detail how we interpret this dynamic specification as a "bound" on permitted behaviors that the controlled system is allowed to retain.

First we note that a trajectory in $\mathcal{H}$ whose associated trace is not in $L(\mathcal{P})$ is definitely impossible in $\mathcal{P}$. Therefore, we must first restrict the specification to behaviors whose associated traces $\mathcal{P}$ is capable of executing. Thus, we first replace $\mathcal{H}$ by

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

We note 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 turn to the language constraint imposed by $\hat{\mathcal{H}}$ on $\mathcal{P}$. To this end we construct the deterministic automaton

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

which we shall employ as our "language specification". Next, 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 $\overline{H}_d := \hat{H}_d \cup \{h_b\}$, with h_b being a new state that we call the *bad* state, and $\overline{\psi}_d$ being 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

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

where $\overline{Q} = Q \times \overline{H}_d$, $\overline{q}_0 = (q_0, \hat{h}_{d0})$, $\overline{Q}_g = Q \times \hat{H}_d$, and where for $\overline{q} = (q, \hat{h}_d) \in \overline{Q}$ and $\sigma \in \Sigma$, $\overline{\delta}(\overline{q}, \sigma)$ is defined as:

$$\overline{\delta}(\overline{q}, \sigma) := \{\overline{q}' = (\delta(q', \sigma) \times \psi(\hat{h}_d, \sigma)) : q' \in \epsilon^*(q)\}.$$

We note at once that $L_m(\overline{\mathcal{P}}) = L(\mathcal{P}) \cap L_m(\overline{\mathcal{H}}_d) = L(\hat{\mathcal{H}})$, and we can readily prove that the trajectory models of $\overline{\mathcal{P}}$ and of $\mathcal{P}$ coincide:

Proposition 3 $\overline{\mathcal{P}} = \mathcal{P}$.

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 least 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 4.

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

In view of Proposition 3, 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 $\hat{\mathcal{H}}$. 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 $\overline{\mathcal{P}}_t$,

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

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 1 *(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 σ 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 1 is stated in the following

Theorem 1 For any path p in $\overline{\mathcal{P}}_t$, p belongs to $\overline{Q}_t$ if and only if $t_p \in \hat{\mathcal{H}}$.

The above theorem shows that we can always translate a dynamic specification into an equivalent static specification.

6 Supervisor synthesis

In the previous section we have seen that the supervisory control problem for a nondeterministic system with dynamic specification can be translated to an equivalent problem with static specifications. We shall thus 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.

To synthesize a supervisor we first lift $\overline{\mathcal{P}}$ to $\tilde{\mathcal{P}}$ using the procedure Extend and define E as in Section 4. If E is not controllable and observable with respect to $L(\tilde{\mathcal{P}})$, then we find the largest sublanguage of E that is controllable and observable (which, in view of the fact that $\Sigma_c \subseteq \Sigma = \Sigma_o$, is its supremal controllable and normal sublanguage) and synthesize a supervisor based on that language. The supervisor design is then carried out off-line in the usual way [10].

An alternate approach for supervisor synthesis is to design a supervisor "on-line". We again lift $\overline{\mathcal{P}}$ to $\tilde{\mathcal{P}}$ but we then proceed somewhat differently: We compute the supremal controllable sublanguage of the legal language E. This can be done with linear complexity for a closed language E. We then design a supervisor on-line using the results of [3]. The resulting supervisor will generate the supremal controllable and normal sublanguage of E.

Finally, we remark that the lifting procedure can actually be bypassed and a direct synthesis approach can be employed, as discussed in detail in [6] [5].

References

[1] M. Fabian and B. Lennartson, 1994. Object oriented supervisory control with a class of nondeterministic specifications, Report No CTH/RT/I-94/007, Chalmers University of Technology, Goteborg, Sweden.

[2] M. Heymann, 1990. Concurrency and discrete-event control. *IEEE Control Systems Magazine, 10(4)*, pp. 103-112.

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

[4] M. Heymann and G. Meyer, 1991. An algebra of discrete-event processes. *NASA Technical Memorandum 102848.*

[5] M. Heymann and F. Lin, 1995. On observability and nondeterminism in discrete event control, *Proceedings of th 33rd Allerton Conference on Communication, Control, and Computing*, pp. 136-145.

[6] M. Heymann and F. Lin, 1996. Discrete event control of nondeterministic discrete event systems, CIS Report No 9601, Technion, Haifa, Israel

[7] M. Heymann and F. Lin, 1996. *Nonblocking supervisory control of nondeterministic systems*, to appear.

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

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

[10] F. Lin and W. M. Wonham, 1988. On observability of discrete event systems. *Information Sciences, 44(3)*, pp. 173-198.

[11] F. Lin and W. M. Wonham, 1994. Supervisory control of timed discrete event systems under partial observation, *IEEE Transactions on Automatic Control, 40(3)*, pp. 558-562.

[12] A. Overkamp, 1994. Supervisory control for nondeterministic systems. *Proccedings of 11th International Conference on Analysis and Optimization of Systems*, pp. 59-65.

[13] A. Overkamp, 1996. Supervisory control using failure semantics and partial specifications, *IEEE Transactions on Automatic Control*, to appear.

[14] P. J. Ramadge and W. M. Wonham, 1989. The control of discrete event systems. *Proceedings of IEEE, 77(1)*, pp. 81-98.

[15] M. Shayman and R. Kumar, 1995. Supervisory control of nondeterministic systems with driven events via prioritized synchronization and trajectory models. SIAM Journal of Control and Optimization, 33(2), pp. 469-497.