
Discrete Event Dynamic Systems: Theory and Applications, 8, 377–429 (1998)

Bisimulation, the Supervisory Control Problem and Strong Model Matching for Finite State Machines

GEORGE BARRETT grbarret@eecs.umich.edu
Department of Electrical Engineering and Computer Science, The University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109–2122

STÉPHANE LAFORTUNE stephane@eecs.umich.edu
Department of Electrical Engineering and Computer Science, The University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109–2122

Abstract. A fundamental relationship between the controllability of a language with respect to another language and a set of uncontrollable events in the Supervisory Control Theory initiated by (Ramadge and Wonham, 1989) and bisimulation of automata models is derived. The theoretical results relating bisimulation to controllability support an efficient solution to the Basic Supervisory Control Problem. Using (Fernandez, 1990) generalization of the partition refinement algorithm of (Paige and Tarjan, 1987), it is possible to find a partition which represents the supremal controllable sublanguage of an automaton with respect to the language of another automaton and a set of events in a worst-case running time of $O(m \log(n))$, where m is the number of transitions and n is the number of states. Utilizing the bisimulation property of language controllability and derived relationships between automata languages and input/output finite-state machine behaviors, a precise relationship is formally derived between Supervisory Control Theory and the system-theoretic problem posed by (DiBenedetto *et al*., 1994) called Strong Input/Output FSM Model Matching. Specifically, it is proven that in deterministic settings instances of each problem can be mapped to the other framework and solved.

Keywords: supervisory control, bisimulation, model matching, controllability

1. Introduction

This paper presents the results of an investigation of the relationship between supervisory control of discrete-event systems (DES) (Ramadge and Wonham, 1989), bisimulation relations (Arnold, 1994, Baeten and Weijland, 1990), and strong input/output finite-state machine model matching (DiBenedetto *et al*., 1994, 1995, 1996). The approach taken throughout this text is from the point of view of automata as DES models (Cassandras *et al*., 1995). A basic understanding of DES supervisory control is assumed, for the introductory material given here is brief. One of the main focal points of this paper is the exploitation of the level of information contained in a finite-state automaton model when solving the supervisory control problem.

Of particular interest is the fact that an automaton model contains not only the language information about a DES but also the branching structure of allowable sequences of events. This level of detail of information allows the supervisory control problem to be generalized to several process-theoretic semantics. A semantics used in computer science which has not previously been applied to the supervisory control problem is the bisimulation semantics. A discussion of bisimulation is given and results are presented for its application to the supervisory control problem.

Supervisory control is not the only system-theoretic problem for which bisimulations can apply. One other such class of problems is strong input/output (I/O) finite-state machine (FSM) model matching introduced in (DiBenedetto *et al*., 1994). This model matching problem is closely related to supervisory control, hence bisimulation is also related to I/O FSM model matching. The other key focus of this paper is to demonstrate how bisimulation can be used to solve both supervisory control problems and model matching problems; thus, an underlying similarity of the two types of problems is exposed.

The contributions of this work are:

1. Theoretical results relating bisimulation and controllability are presented in Section 3 which support an efficient algorithm for solving the "Basic Supervisory Control Problem." The algorithm, presented in Section 4, is based on a partition refinement algorithm which finds the coarsest relation partition with complexity $O(m \log(n))$ where m is the number of related pairs in the relation and n is the number of elements in the partition. Understanding how bisimulation, as the descriptive semantics, relates to supervisory control lends insight toward utilizing weaker semantics, e.g., trajectory model (Heymann and Meyer, 1991), failure, failure trace, ready, ready trace, and trace (language) (Baeten and Weijland, 1990).

2. Section 5 reviews I/O FSM model matching as presented in (DiBenedetto *et al*., 1994, 1995, 1996). In addition, we present a precise relationship between "behavioral equivalence" of FSMs in the I/O FSM model matching framework and bisimulation between two FSMs.

3. The *behavior* of I/O FSMs is related to the language properties of associated Moore automata in section 7.1. It is shown that, in deterministic settings, the notion of *behavioral inclusion* of I/O FSMs presented by (DiBenedetto *et al*., 1995, 1996) is equivalent to the inclusion of families of marked languages represented by Moore automata.

4. It is also shown in Section 7 that, in deterministic settings, I/O FSM behaviors that are achievable in the presence of measurable disturbances are precisely those whose associated set of marked languages is controllable.

5. In Section 7.2 a method is presented for solving the "Basic Supervisory Control Problem" by mapping it to an instance of the "Strong I/O FSM Model Matching with Measurable Disturbances to A Maximal Set of Reference Behaviors Problem." (DiBenedetto *et al*., 1996) give examples of solving the BSCP in their I/O FSM framework; however, they do not present a formal proof that the supremal controllable sublanguage is obtained. We establish, through formal proof, an exact relationship between the two paradigms.

6. A method is presented in Section 7.3 for solving the "Strong I/O FSM Model Matching with Measurable Disturbances to A Maximal Set of Reference Behaviors Problem" of (DiBenedetto *et al*., 1995 and 1996) in deterministic settings by mapping it to an instance of the "Basic Supervisory Control Problem".

This paper is organized as follows. Section 2 gives a brief introduction to results from formal language theory, DES, the Basic Supervisory Control Problem, and bisimulation. An efficient algorithm for finding bisimulation relations is also discussed. Section 3 discusses the relationship between controllability in the supervisory control problem and bisimulation. Section 4 discusses using bisimulation to solve the Basic Supervisory Control Problem with both off-line and on-line strategies. Section 5 discusses I/O finite-state machine model matching, its operators and some of its associated problems. The relationship between I/O FSM model matching and supervisory control is discussed in Sections 6 and 7. Section 6 presents methods of discrete-event model conversion so the two paradigms may be compared, and Section 7 formally establishes a link between Strong I/O FSM Model Matching and Supervisory Control Theory. The paper concludes with Section 8 where the basic ideas are summarized. Parts of this paper, in its preliminary form, have appeared in (Barrett and Lafortune, 1996, 1997).[1]

2. Preliminaries

2.1. Automata and Languages

For the investigation and discussion of discrete-event systems at a logical level of abstraction, the automaton will be used as the primary descriptive model.

DEFINITION 2.1 *(Cassandras et al., 1995)*[2] *A* **deterministic finite-state automaton (DFSA)**, *denoted G, is a six-tuple*

$$G = (X_G, \Sigma_G, \delta_G, act_G, x_{G0}, X_{Gm})$$

that generates the language $\mathcal{L}(G)$ and marks the language $\mathcal{L}_m(G)$ where

$$\begin{aligned}
X_G &= \textit{finite set of states of } G \\
\Sigma_G &= \textit{set of events associated with the transitions in } G \\
\delta_G &= \textit{partial transition function of } G,\ \delta_G : X_G \times \Sigma_G \to X_G \\
act_G(x) &= \textit{active event set of } G \textit{ at state } x \in X_G, \textit{ i.e., subset of } \Sigma_G \\
&\quad \textit{for which } \delta_G(x, \cdot) \textit{ is defined} \\
x_{G0} &= \textit{initial state of } G \\
X_{Gm} &= \textit{a subset of } X_G \textit{ which represent} \text{ marked } \textit{states.}
\end{aligned}$$

The transition function δ_G is not necessarily total as in the "standard" definition of automaton, and all automata in this paper are assumed to be accessible, i.e., all unreachable states have been deleted from the model. If X_{Gm} is omitted, then it is understood that $X_{Gm} = X_G$. Furthermore, it is common to define the empty trace ϵ to be the trace containing no events. The transition function, δ_G, is often extended from events to traces recursively:

$$\delta_G(x_G, \epsilon) = x_G,$$

and for $t \in \Sigma_G^*$, $\sigma \in \Sigma_G$:

$$\delta_G(x_G, t\sigma) = \delta_G(\delta_G(x_G, t), \sigma).$$

If G is a nondeterministic finite-state automaton, then $\delta_G(x_G, t)$ is a set. The notion of language mentioned above is now formally defined.

DEFINITION 2.2 *(Cassandras et al., 1995) The language* **generated** *by G is:*

$$\mathcal{L}(G) = \{t \in \Sigma_G^* : \delta_G(x_{G0}, t) \textit{ is defined}\}.$$

Likewise, the special subset of $\mathcal{L}(G)$ that represents the **marked** *language of G is defined as:*

$$\mathcal{L}_m(G) = \{t \in \mathcal{L}(G) : \delta_G(x_{G0}, t) \in X_{Gm}\}.$$

The notion of a set of marked states can be generalized to several sets of marked states $\{X_{m,1}, \ldots, X_{m,k}\}$. This generalization is discussed as *parameterized states* in (Arnold, 1994); thus, a single automaton can be viewed as marking several languages. This idea of multiple marking sets is useful when states are considered to have associated "outputs" as is the case of Moore type automata; in this case, it is possible to associate an output value with a set of specifically marked states. Such a generalization was first used in the context of supervisory control in (Thistle *et al*., 1995).

Two common operations on automata are the product, denoted $\times$, and the parallel composition, denoted $||$. These two operations are designed to describe the interactions between discrete-event systems modeled as automata.

DEFINITION 2.3 *(Cassandras et al., 1995) The* **product** *of two automata G_1 and G_2 where:*

$$G_1 = (X_{G1}, \Sigma_{G1}, \delta_{G1}, act_{G1}, x_{G01}, X_{Gm1})$$

$$G_2 = (X_{G2}, \Sigma_{G2}, \delta_{G2}, act_{G2}, x_{G02}, X_{Gm2})$$

is

$$G_1 \times G_2 = (X_{G1} \times X_{G2}, \Sigma_{G1} \cap \Sigma_{G2}, \delta, act_{G_{1\times 2}}, (x_{G01}, x_{G02}), X_{Gm1} \times X_{Gm2})$$

where

$$\delta((x_{G1}, x_{G2}), \sigma) = \begin{cases} (\delta_{G1}(x_{G1}, \sigma), \delta_{G2}(x_{G2}, \sigma)) & \textit{if } \sigma \in act_{G1}(x_{G1}) \cap act_{G2}(x_{G2}) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

and

$$act_{G_{1\times 2}}(x_{G1}, x_{G2}) = act_{G1}(x_{G1}) \cap act_{G2}(x_{G2}).$$

A resulting composite state is considered to be marked if and only if both constituent states are marked. Intuitively, the product machine represents the set of possible actions that are common to both machines G_1 and G_2, hence $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$ and $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$.

DEFINITION 2.4 *(Cassandras et al., 1995) The* **parallel composition** *of two automata G_1 and G_2 is*

$$G_1||G_2 = (X_{G1} \times X_{G2}, \Sigma_{G1} \cup \Sigma_{G2}, \delta, act_{G_{1||2}}, (x_{G01}, x_{G02}), X_{Gm1} \times X_{Gm2})$$

where

$$\delta((x_{G1}, x_{G2}), \sigma) = \begin{cases} (\delta_{G1}(x_{G1}, \sigma), \delta_{G2}(x_{G2}, \sigma)) & \text{if } \sigma \in act_{G1}(x_{G1}) \cap act_{G2}(x_{G2}) \\ (\delta_{G1}(x_{G1}, \sigma), x_{G2}) & \text{if } \sigma \in act_{G1}(x_{G1}) \backslash \Sigma_{G2} \\ (x_{G1}, \delta_{G2}(x_{G2}, \sigma)) & \text{if } \sigma \in act_{G2}(x_{G2}) \backslash \Sigma_{G1} \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

and

$$act_{G_{1||2}}(x_{G1}, x_{G2}) = [act_{G1}(x_{G1}) \cap act_{G2}(x_{G2})] \cup [act_{G1}(x_{G1}) \backslash \Sigma_{G2}] \cup [act_{G2}(x_{G2}) \backslash \Sigma_{G1}].$$

With the parallel composition, only events common to the alphabets of both automata must be synchronized on, and as in the product a resulting composite state is considered to be marked if and only if both constituent states are marked.

In addition to the above operations, it is convenient to define the natural projection operator which "removes" events from a trace which do not belong to a particular event set.

DEFINITION 2.5 *(Cassandras et al., 1995) The* **natural projections** *$P_{\Sigma_i} : (\Sigma_1 \cup \Sigma_2)^* \longrightarrow \Sigma_i^*$ for $i = 1, 2$ are*

$$\begin{aligned} P_{\Sigma_i}(\epsilon) &:= \epsilon \\ P_{\Sigma_i}(\sigma) &:= \begin{cases} \sigma & \text{if } \sigma \in \Sigma_i \\ \epsilon & \text{if } \sigma \notin \Sigma_i \end{cases} \\ P_{\Sigma_i}(s\sigma) &:= P_{\Sigma_i}(s) P_{\Sigma_i}(\sigma) \text{ for } s \in (\Sigma_1 \cup \Sigma_2)^*, \sigma \in (\Sigma_1 \cup \Sigma_2), \end{aligned}$$

and the corresponding inverse maps $P_{\Sigma_i}^{-1} : \Sigma_i^* \longrightarrow 2^{(\Sigma_1 \cup \Sigma_2)^*}$ are

$$P_{\Sigma_i}^{-1}(t) := \{s \in (\Sigma_1 \cup \Sigma_2)^* : P_{\Sigma_i}(s) = t\}.$$

The definitions of $P(\cdot)$ and $P^{-1}(\cdot)$ are extended to sets in the usual manner. Given the definition of inverse projection, the result of the parallel composition can be written as:

$$\begin{aligned} \mathcal{L}(G_1||G_2) &= P_{\Sigma_1}^{-1}[\mathcal{L}(G_1)] \cap P_{\Sigma_2}^{-1}[\mathcal{L}(G_2)] \\ \mathcal{L}_m(G_1||G_2) &= P_{\Sigma_1}^{-1}[\mathcal{L}_m(G_1)] \cap P_{\Sigma_2}^{-1}[\mathcal{L}_m(G_2)]. \end{aligned}$$

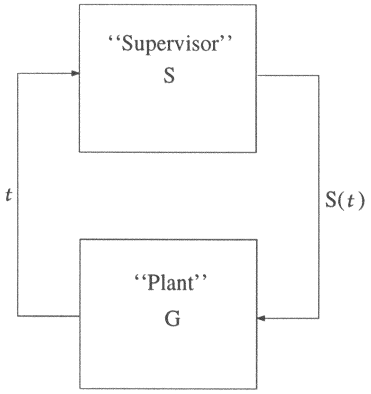

Figure 1. Feedback loop of supervisory control.

It is also convenient to consider an automaton which represents the projected language of G_1. This "projected" automaton, $P(G_1)$, is denoted here using the same notation used for language projection, i.e., for some set of events Σ_{arb}, $\mathcal{L}(P_{\Sigma_{arb}}(G_1)) = P_{\Sigma_{arb}}(\mathcal{L}(G_1))$. Because the projected automaton, as it is used here, only needs to capture the projected language aspects of G_1, it will be assumed that $P_{\Sigma_{arb}}(G_1)$ is a DFSA. If uniqueness of $P_{\Sigma_{arb}}(G_1)$ is required, then it can be assumed that $P_{\Sigma_{arb}}(G_1)$ is its minimum-state representation.[3] The inverse projection of an automaton can be defined in a similar manner.

Finally, when considering multiple automata, it is common that not every event is defined in every automaton model. For this reason, the symbol Σ (with no subscripts) is used to denote the universe of events for all automata of interest.

2.2. The Basic Supervisory Control Problem

Supervisory Control Theory (SCT) deals with the control of discrete-event systems. A point of view is assumed that some behavior of a plant modeled as a DES is *illegal* and must be disabled by a controller called a *supervisor* (Ramadge and Wonham, 1987, 1989). This control scheme is depicted in Figure 1. Associated with G is a set of uncontrollable events, $\Sigma_{uc} \subseteq \Sigma_G$ which cannot be directly disabled by control; hence, the supervisor is a function

$$S : \mathcal{L}(G) \longrightarrow \{\gamma \in 2^{\Sigma} : \Sigma_{uc} \subseteq \gamma\}.$$

The set of enabled events that G can execute after a trace t is given by

$$act_{S/G}(t) = S(t) \cap act_G(\delta_G(x_0, t)). \tag{1}$$

Note that S/G does not imply quotient in this usage; rather, S/G means G controlled by S as described in Eqn. 1. Furthermore, $\mathcal{L}(S/G)$ is the generated language of the closed-loop

system under the control of S, and $\mathcal{L}_m(S/G) = \mathcal{L}(S/G) \cap \mathcal{L}_m(G)$. Consider a sublanguage K of $\mathcal{L}(G)$ which is the desired set of traces for the controlled discrete-event system. Let K be represented by the automaton $H = (X_H, \Sigma_H, \delta_H, act_H, x_{H0}, X_{Hm})$, i.e., $K = \mathcal{L}_m(H)$. The language K is said to be *controllable* (with respect to $\mathcal{L}(G)$ and Σ_{uc}) if[4]

$$\overline{K}\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}. \tag{2}$$

This relationship between K and $\mathcal{L}(G)$ can be interpreted several ways. At the language level, Eqn. 2 says that for a desired language to be controllable the execution of an uncontrollable event must not generate a trace that is not in K. Because it is assumed here that K is represented by the automaton H, controllability can also be interpreted at the state level. Equation 2 states that for K to be controllable then for every state pair (x_H, x_G) in the product automaton $H \times G$ the condition $P_{\Sigma_{uc}}(act_G(x_G)) \subseteq P_{\Sigma_{uc}}(act_H(x_H))$ must be satisfied where $P_{\Sigma_{uc}}(\cdot)$ represents the natural projection onto the set of uncontrollable events. Furthermore, if it is given that $K \subseteq \mathcal{L}(G)$, then the condition becomes an equality. These conditions indicate that controllability can be viewed as a relation on the set of states of $H \times G$. This state interpretation is explored later in Section 3.

DEFINITION 2.6 **Basic Supervisory Control Problem (BSCP)** *(Ramadge and Wonham, 1987) Given a DES modeled by DFSA G, $\Sigma_{uc} \subseteq \Sigma_G$, and desired legal language, $K = \overline{K} \subseteq \mathcal{L}(G)$, build supervisor S such that:*

1. $\mathcal{L}(S/G) \subseteq K$
2. *For any other S' such that $\mathcal{L}(S'/G) \subseteq K$, $\mathcal{L}(S'/G) \subseteq \mathcal{L}(S/G)$, i.e., $\mathcal{L}(S/G)$ is as large as possible.*

The solution to the BSCP is called the *minimally restrictive solution* (MRS): $\mathcal{L}(S/G) = K^{\uparrow}$, where $K^{\uparrow}$ is the supremal controllable sublanguage of K with respect to $\mathcal{L}(G)$ and Σ_{uc} (Wonham and Ramadge, 1987).

A common solution technique, the "standard" algorithm (Wonham and Ramadge, 1987), for the BSCP is to form the automaton product $H \times G$ and iteratively remove states which violate the controllability condition or are not reachable. If n_H and n_G are the number of states in H and G respectively, then the worst-case running time of this type of algorithm is $O(|\Sigma| n_H n_G)$ using the technique in (Kumar *et al.*, 1991) for cleverly pruning the state space or using the constructive approach in (Hadj-Alouane *et al.*, 1994).[5] More "efficient"[6] algorithms exist which are based on an interesting relation between controllability and bisimulation.

Because the blocking behavior of a system is of interest, we define a *nonblocking supervisor* as one which allows the closed loop system to complete any trace in its generated language to a marked trace.

DEFINITION 2.7 **Nonblocking Version of BSCP (BSCP–NB)** *(Ramadge and Wonham, 1987) Given DES G, $\Sigma_{uc} \subseteq \Sigma_G$, and desired legal marked language, $K \subseteq \mathcal{L}_m(G)$, with K*

assumed to be $\mathcal{L}_m(G)$-closed, build nonblocking *supervisor S such that:*

1. $\mathcal{L}_m(S/G) \subseteq K$
2. *For any other nonblocking S' such that $\mathcal{L}_m(S'/G) \subseteq K$, $\mathcal{L}(S'/G) \subseteq \mathcal{L}(S/G)$, i.e., $\mathcal{L}_m(S/G)$ is as large as possible.*

The solution to the BSCP–NB is called the *minimally restrictive nonblocking solution* (MRNBS): $\mathcal{L}(S/G) = \overline{K^{\uparrow}}$ and $\mathcal{L}_m(S/G) = K^{\uparrow}$.

The standard method for calculating $K^{\uparrow}$ is by an iterative algorithm that starts with an automaton H that marks K. The product automaton, $H \times G$, is formed as in the BSCP, and the resulting automaton is pruned in an iterative manner until convergence. The iterative procedure consists of (i) deleting states that violate the controllability condition, and (ii) deleting states that are not accessible or coaccessible, i.e., "trimming". Because trimming does not perserve controllability, removing states which violate controllability must be performed following each trim operation. The complexity of trimming is $O(n_H n_G)$; hence if K is not prefix-closed, the worst-case running time of this algorithm is $O(|\Sigma|(n_H n_G)^2)$.

2.3. Bisimulation Relations, Partitions and the Bisimulation Semantics

In what follows, $x \xrightarrow{\sigma} x'$ denotes that there exists a transition with the label σ from state x to state x'.

DEFINITION 2.8 *Let H and G be as in Subsection 2.2. A* **bisimulation relation** *of H and G with respect to $\Sigma_B \subseteq \Sigma_H \cup \Sigma_G$ is a binary relation $\psi \subseteq X_H \times X_G$ satisfying (Arnold, 1994, Baeten and Weijland, 1990):*

1. *If $\psi(x_H, x_G)$, $\sigma \in \Sigma_B$ and $x_H \xrightarrow{\sigma} x'_H$, then there is a x'_G such that $x_G \xrightarrow{\sigma} x'_G$ and $\psi(x'_H, x'_G)$.*
2. *If $\psi(x_H, x_G)$, $\sigma \in \Sigma_B$ and $x_G \xrightarrow{\sigma} x'_G$, then there is a x'_H such that $x_H \xrightarrow{\sigma} x'_H$ and $\psi(x'_H, x'_G)$.*
3. *(Bisimulation with marking) $\psi(x_H, x_G)$ implies $x_H \in X_{Hm}$ iff $x_G \in X_{Gm}$.*

For fixed parameter Σ_B, bisimulation relations are closed under arbitrary unions (Arnold, 1994, Baeten and Weijland, 1990), so there exists a *greatest* bisimulation relation.[7] Automata H and G are *bisimilar* with respect to Σ_B, denoted $H \underline{\leftrightarrow}_{\Sigma_B} G$, if there exists a bisimulation relation of H and G with respect to Σ_B such that each state of H and each state of G appears in the relation and (x_{H0}, x_{G0}) is in the relation.

For example, it is easily verified that F_1 and F_2 of Figure 2 are bisimilar with respect to the event "a". The greatest bisimulation relation of F_1 and F_2 with respect to "a", $\Phi_a \subseteq X_{F1} \times X_{F2}$, is easily verified to be:

$$\Phi_a = \{(A,1),(B,2),(B,3),(B,4),(C,2),(C,3),(C,4),(D,2),(D,3),(D,4),(E,2),(E,3),(E,4)\}.$$

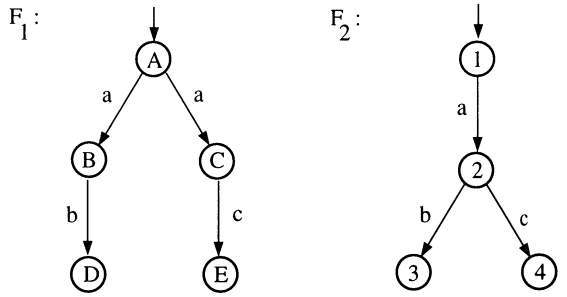

Figure 2. Example of simulation relation.

On the other hand, F_1 and F_2 are not bisimiliar with respect to the total event set $\Sigma = \{a, b, c\}$; for instance, after "a" occurs F_1 can be at a state where "c" can never occur, but "c" can always occur following "a" in F_2; hence, $F_1 \not\approx_\Sigma F2$.

DEFINITION 2.9 *Let H and G be as in Subsection 2.2. A* **simulation relation** *of H and G with respect to $\Sigma_S \subseteq \Sigma_H$ is a binary relation $\psi \subseteq X_H \times X_G$ satisfying (Arnold, 1994):*

1. *If $\psi(x_H, x_G)$, $\sigma \in \Sigma_S$ and $x_H \xrightarrow{\sigma} x'_H$, then there is a x'_G such that $x_G \xrightarrow{\sigma} x'_G$ and $\psi(x'_H, x'_G)$.*
2. *(Simulation with marking) $\psi(x_H, x_G)$ implies $x_H \in X_{Hm}$ iff $x_G \in X_{Gm}$.*

If ψ is a simulation relation and $\psi(x_H, x_G)$ then x_G *simulates* x_H. Automaton H is *simulated* by automaton G with respect to Σ_S, denoted $H \sqsubseteq_{\Sigma_S} G$, if there exists a simulation relation between H and G with respect to Σ_S such that every state of H appears in the relation and (x_{H0}, x_{G0}) is in the relation (if $\Sigma_S = \Sigma_H$ then we will drop the subscript on $\sqsubseteq$.) Clearly, if H and G are deterministic, then $H \sqsubseteq G$ iff $\mathcal{L}(H) \subseteq \mathcal{L}(G)$ and $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G)$.

As an example of simulation relation, consider again Figure 2 with $\Sigma = \{a, b, c\}$. Automaton F_1 is simulated by automaton F_2 with respect to Σ; however, F_2 is not simulated by F_1 with respect to Σ (by the same argument given above for $F_1 \not\approx_\Sigma F_2$). Notice that $\mathcal{L}(F_1) = \mathcal{L}(F_2)$. This example also shows the inability of language to capture nondeterministic behavior described by the branching structure of finite-state automata.

It is often convenient to consider a bisimulation relation between an automaton and itself. The greatest bisimulation relation of an automaton with itself (with respect to some event set Σ_A) is called the automaton's *greatest autobisimulation* (Baeten and Weijland, 1990) (with respect to Σ_A). Observe that the greatest autobisimulation relation of an automaton is an equivalence relation on the states of the automaton.

DEFINITION 2.10 *Let G be a finite-state automaton (not necessarily deterministic) with greatest autobisimulation relation Φ (with respect to Σ_G.) Denote by $\mathcal{C}$ the set of equivalence*

classes induced by Φ. *The* **normal form** *of* G, *denoted by* $N(G)$, *is the automaton generated by using the equivalence classes of* Φ *as the states of* $N(G)$ *(Baeten and Weijland, 1990). The states of* $N(G)$ *inherit marking information and all incoming and outgoing event transitions from the previous states of* G. $N(G)$ *can be called the minimum state realization of* G.

In the trace or language semantics, two discrete-event systems modeled by DFSA A_1 and A_2, are considered equivalent if they have equivalent trace sets, i.e., $\mathcal{L}(A_1) = \mathcal{L}(A_2)$ and $\mathcal{L}_m(A_1) = \mathcal{L}_m(A_2)$. The bisimulation semantics, however, has a more detailed requirement for DES equivalence. Within the bisimulation semantics two DES modeled by automata A_1 and A_2, are equivalent if their *normal forms* are equal (isomorphic), i.e., $N(A_1) = N(A_2)$ (Baeten and Weijland, 1990). Figure 2 shows a case where two automata are equivalent in the trace semantics but not equivalent in the bisimulation semantics. If A_1 and A_2 are both deterministic, then $N(A_1) = N(A_2)$ iff $\mathcal{L}(A_1) = \mathcal{L}(A_2)$ and $\mathcal{L}_m(A_1) = \mathcal{L}_m(A_2)$.

Bisimulation is a more detailed semantics than the trajectory model semantics in (Heymann and Meyer, 1991) and ready (accepting trace (Arnold, 1994)), ready trace, failure (refusing trace (Arnold, 1994)), failure trace, and trace semantics in (Baeten and Weijland, 1990), that is, bisimulation equivalence implies equivalence in all of the above listed semantics. Indeed, bisimulation is *too* detailed (Bloom *et al*., 1988) for many purposes, i.e., it distinguishes DES that are indistinguishable given the observation of a sequence of events. Referring to Figure 2 again, upon observing the events "ab" a supervisor could not distinguish F_1 from F_2 without receiving *menus*[8] of possible events from the DES at each state along the path ab. Despite bisimulation being finer than trace equivalence, algorithms exist for bisimulation equivalence that are more efficient than analogous algorithms for regular languages. If the automata being processed by these algorithms are deterministic, then the bisimulation results are equivalent to trace results.

2.4. Using Partition Refinement to Find Bisimulation Relations

As previously stated in Section 2.3, the greatest autobisimulation relation of an automaton (with respect to Σ_A) is an equivalence relation making it representable as a partition of the states of the automaton. Furthermore, it has been shown that the problem of finding the greatest autobisimulation relation of an automaton with respect to an event set Σ_A is equivalent to the *coarsest relational partition* problem (Fernandez, 1990, Paige and Tarjan, 1987).

The algorithm of (Paige and Tarjan, 1987) for solving the coarsest relational partition problem considers the case of a single relation, i.e., $|\Sigma_A| = 1$. (Fernandez, 1990) generalizes the algorithm of Paige and Tarjan to handle the case when $|\Sigma_A| \geq 1$. When applied to an automaton, A, the algorithm of Fernandez runs in time bounded by[9] $O(m \log(n))$ and space bounded by $O(m + n)$ where m is the number of transitions in A, and n is the number of states in A.

The greatest bisimulation relation of two automata H and G with respect to Σ_A, denoted Φ_{Σ_A}, can be found by first forming the autobisimulation (with respect to Σ_A) of the union of H and G and then considering only the ordered state-pairs of the form (x_H, x_G). As

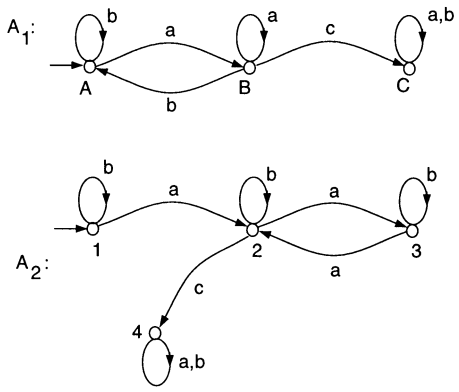

Figure 3. Example automata.

mentioned above, the greatest autobisimulation can be found by solving the coarsest relational partition problem; hence, it is in this manner that the coarsest relational partition induces the greatest bisimulation relation Φ_{Σ_A}. To illustrate this fact, the following subsection presents an example. The computation of Π has worst-case time complexity of $O(|\Sigma_A|(n_H + n_G)\log(n_H + n_G))$ where n_H and n_G are the number of states in H and G respectively.

2.4.1. Example of Finding Coarsest Stable Relational Partition

The details of the efficient algorithm can be found in (Fernandez, 1990). Here, a simple example of a "naïve" implementation of the algorithm in tabular form is presented.

Consider the problem of finding a bisimulation relation between the two automata, A_1 and A_2, shown in Figure 3. States are shown in capital letters or numbers, while transitions are shown by lower case letters. Blocks in a partitioned set are denoted by Roman numerals.

Begin by forming a block, I, of all of the states of A_1 and A_2, and put this block into a set, Π_0:

$$\Pi_0 = \{\mathrm{I}\} = \{\{A, B, C, 1, 2, 3, 4\}\}$$

The basic iteration on Π_i involves choosing a block, j, in the partition and determining its preimage set

$$E^{-1}_{[\sigma]}(j) = \{x | \exists x' \in j \text{ such that } x \xrightarrow{\sigma} x'\}.$$

Each block of the partition is examined and *split* using preimage sets. The algorithm terminates when $\Pi_i = \Pi_{i-1}$. For example, the transition relation of Figure 3 can be expressed

in tabular form showing from each state which block is reachable for a given event label:

	a	b	c
A	I	I	Ø
B	I	I	I
C	I	I	Ø
1	I	I	Ø
2	I	I	I
3	I	I	Ø
4	I	I	Ø

This tabular form allows the preimage sets to be determined by inspection: group all states with identical transitions. Notice that the block I is not stable with respect to itself, so it splits. The refined partition is (here, block names are being reused):

$$\Pi_1 = \{\text{I,II}\} = \{\{A, C, 1, 3, 4\}, \{B, 2\}\}.$$

Repeating this process yields the tabular transition relation:

	a	b	c
A	II	I	Ø
C	I	I	Ø
1	II	I	Ø
3	II	I	Ø
4	I	I	Ø
B	II	I	I
2	I	II	I

In the terminology of (Paige and Tarjan, 1987) block II is a *splitter* of blocks I and II. In this case, block I is also a splitter. (Splitters are common to both the naïve algorithm shown here and the efficient algorithms in (Fernandez, 1990, Paige and Tarjan, 1987). The key to the efficiency of the improved algorithms is the intelligent chosing of splitters.) Because the partition contains blocks that can be split by the splitters, the partition is not *stable* with respect to the splitters. Forming groups with similar transitions yields the partition:

$$\Pi_2 = \{\text{I,II,III,IV}\} = \{\{A, 1, 3\}, \{C, 4\}, \{B\}, \{2\}\}.$$

Notice, blocks III and IV are singletons, so they cannot be split; thus, they need not be examined in the tabular transition relation:

	a	b	c
A	III	I	Ø
1	IV	I	Ø
3	IV	I	Ø
C	II	II	Ø
4	II	II	Ø

Examining this table yields:

$$\Pi_3 = \{\text{I,II,III,IV,V}\} = \{\{A\}, \{1, 3\}, \{C, 4\}, \{B\}, \{2\}\},$$

where only one block has elements from A_1 and A_2, namely (C,4). Looking at the transition behavior of these two states yields:

	a	b	c
C	III	III	Ø
4	III	III	Ø

Hence Π_3 is the coarsest partition of $X_{A1} \cup X_{A2}$ that is stable with respect to $\{a, b, c\}$ transitions. This partition is equivalent to the greatest autobisimulation relation on $X_{A1} \cup X_{A2}$, namely the greatest bisimulation relation, $\mathcal{B}_{auto}$, on $[X_{A1} \cup X_{A2}] \times [X_{A1} \cup X_{A2}]$, or

$$\mathcal{B}_{auto} = \{(A, A), (1, 1), (1, 3), (3, 1), (3, 3), (C, C), (C, 4), (4, C), (4, 4), (B, B), (2, 2)\}.$$

The greatest bisimulation relation of A_1 and A_2, Φ_{Π_3}, can be taken as the subset of $\mathcal{B}_{auto}$ that is in $X_{A1} \times X_{A2}$, specifically:

$$\Phi_{\Pi_3} = \{(C, 4)\}.$$

The greatest bisimulation relation, Φ_{Π_3}, found in this subsection does not contain all states of A_1 and A_2 (or even the initial states) so it can be concluded that A_1 and A_2 are not bisimilar.

3. Controllability and Bisimulation

This section presents the main theoretical results of this paper that relate controllability with bisimulation. These results support efficient algorithms that will be discussed later for solving supervisory control and related problems.

THEOREM 3.1 *Given two deterministic finite-state automata $A_1 = (X_{A1}, \Sigma_{A1}, \delta_{A1}, act_{A1}, x_{A1,0})$ and $A_2 = (X_{A2}, \Sigma_{A2}, \delta_{A2}, act_{A2}, x_{A2,0})$ where $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, let $\psi \subseteq X_{A1} \times X_{A2}$ be the set of state pairs that are reachable by traces in $\mathcal{L}(A_1)$. Let $\Sigma_{uc} \subseteq \Sigma_{A2}$ be the set of uncontrollable events.*
Then $\mathcal{L}(A_1)$ is controllable with respect to $\mathcal{L}(A_2)$ and Σ_{uc} if and only if ψ is a bisimulation relation of A_1 and A_2 with respect to Σ_{uc}.

Proof 3.1: For deterministic automata, bisimulation is simply two-way simulation; hence we need only show $\mathcal{L}(A_1)\Sigma_{uc} \cap \mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$ iff A_1 is simulated by A_2 with respect to Σ_{uc}, and A_2 is simulated by A_1 with respect to Σ_{uc}. The language inclusion assumption gives half of this requirement because for deterministic automata $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ iff $A_1 \sqsubseteq A_2$; hence A_1 is simulated by A_2 with respect to Σ_{uc}.

The burden of the proof rests upon showing that given $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$ we have $\mathcal{L}(A_1)\Sigma_{uc} \cap \mathcal{L}(A_2) \subseteq \mathcal{L}(A_1)$ iff A_2 is simulated by A_1 with respect to Σ_{uc}.

Using contraposition for showing simulation of A_2 by A_1 with respect to the uncontrollable events yields:

given $\mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$, then

$$\begin{aligned}
\mathcal{L}(A_1)\Sigma_{uc} \cap \mathcal{L}(A_2) \not\subseteq \mathcal{L}(A_1) &\Leftrightarrow \exists t \in \mathcal{L}(A_1), \exists \sigma_{uc} \in \Sigma_{uc} \text{ such that } t\sigma_{uc} \notin \mathcal{L}(A_1) \wedge t\sigma_{uc} \in \mathcal{L}(A_2).\\
&\Leftrightarrow \exists x_{A1} = \delta_{A1}(x_{A1,0}, t), \exists x_{A2} = \delta_{A2}(x_{A2,0}, t), \exists \sigma_{uc} \in \Sigma_{uc} \text{ such that } \sigma_{uc} \notin act_{A1}(x_{A1}) \text{ and } \sigma_{uc} \in act_{A2}(x_{A2}).\\
&\Leftrightarrow \exists x_{A1} = \delta_{A1}(x_{A1,0}, t), \exists x_{A2} = \delta_{A2}(x_{A2,0}, t) \text{ such that } \Sigma_{uc} \cap act_{A2}(x_{A2}) \not\subseteq \Sigma_{uc} \cap act_{A1}(x_{A1}).\\
&\Leftrightarrow \exists x_{A1} = \delta_{A1}(x_{A1,0}, t), \exists x_{A2} = \delta_{A2}(x_{A2,0}, t) \text{ such that } x_{A2} \text{ is not simulated by } x_{A1} \text{ with respect to } \Sigma_{uc}.\\
&\Leftrightarrow A_2 \text{ is not simulated by } A_1 \text{ with respect to } \Sigma_{uc}.
\end{aligned}$$

■

The greatest bisimulation relation, Φ_{uc}, between two automata with respect to the set of uncontrollable events may be *too* large, that is, it may contain unreachable state-pairs. The importance of ψ is derived from the need to be able to determine if the bisimulation is violated by reachable state pairs or if the bisimulation is violated by unreachable state pairs. In the latter case, the violation is not of consequence.

Theorem 3.1 requires both automata to be deterministic. Generally, for nondeterministic automata, bisimulation is much too strong a requirement for language controllability. That is, bisimulation with respect to the set of uncontrollable events is, in general, sufficient for language controllability, but it is not necessary.

DEFINITION 3.1 *Let a DES be modeled by the DFSA* $G = (X_G, \Sigma_G, \delta_G, act_G, x_{G0})$*, and let a desired behavior be modeled by the DFSA* $H = (X_H, \Sigma_H, \delta_H, act_H, x_{H0})$*. Denote by* $[H \times G]^{\uparrow}$ *the automaton formed using the standard algorithm (Wonham and Ramadge, 1987) for computing the supremal controllable sublanguage of* $\mathcal{L}(H)$ *with respect to* $\mathcal{L}(G)$ *and* Σ_{uc}*, i.e., forming the product* $H \times G$ *and iteratively pruning states which 1) violate the controllability condition or 2) are unreachable; hence,* $\mathcal{L}([H \times G]^{\uparrow}) = \mathcal{L}(H \times G)^{\uparrow}$.

Theorem 3.1 implies that for two automata H and G as in Definition 3.1 where $\mathcal{L}(H) \subseteq \mathcal{L}(G)$, $\mathcal{L}(H)$ is controllable with respect to $\mathcal{L}(G)$ and Σ_{uc} if and only if the set of all reachable state- pairs of $H \times G$ is a bisimulation relation of H and G with respect to Σ_{uc}. The following theorem characterizes the supremal controllable sublanguage of $\mathcal{L}(H)$ with respect to $\mathcal{L}(G)$ and Σ_{uc}.

THEOREM 3.2 *Let* G *and* H *be as in Definition 3.1 such that* $\mathcal{L}(H) \subseteq \mathcal{L}(G)$ *and* $\mathcal{L}(H)^{\uparrow} = \mathcal{L}([H \times G]^{\uparrow})$*. Denote the (accessible) state space of* $[H \times G]^{\uparrow}$ *by* $S^{\uparrow}$*. Let the set of*

uncontrollable events be Σ_{uc}, and let Φ_{uc} be the greatest bisimulation relation between H and G with respect to Σ_{uc}. Then

1. $(x_{H0}, x_{G0}) \in S^{\uparrow}$ *iff* $\Phi_{uc}(x_{H0}, x_{G0})$, *and*

2. $(x_H, x_G) \in S^{\uparrow}$ *iff* $\Phi_{uc}(x_H, x_G)$ *and* (x_H, x_G) *is reachable from* (x_{H0}, x_{G0}) *by a sequence of state transitions that never leave* Φ_{uc}.

Proof 3.2. Implications 1 and 2 are proved seperately.

1. Follows directly from Theorem 3.1. The existence of a nonempty controllable sublanguage mandates the existence of a bisimulation relation with respect to Σ_{uc} that contains (x_{H0}, x_{G0}) and vice-versa.

2. Let $R = (X_R, \Sigma_R, \delta_R, act_R, x_{R0})$ where

$$\begin{aligned} X_R &= \{x_R \in X_H \times X_G | x_H \in X_H, x_G \in X_G, x_R = (x_H, x_G) \in \Phi_{uc}\} \\ x_{R0} &= (x_{H0}, x_{G0}) \\ \forall \sigma \in \Sigma &: \\ \delta_R((x_H, x_G), \sigma) &= \begin{cases} (\delta_H(x_H, \sigma), \delta_G(x_G, \sigma)) & \text{if } (\delta_H(x_H, \sigma), \delta_G(x_G, \sigma)) \in X_R \\ \text{undefined} & \text{otherwise} \end{cases} \\ act_R((x_H, x_G)) &= \{\sigma \in \Sigma | \delta_R((x_H, x_G), \sigma) \text{ is defined}\}. \end{aligned}$$

   Define $X_R^{\uparrow}$ to be the reachable state space of R; hence, $X_R^{\uparrow}$ is the greatest bisimulation relation between H and G with respect to Σ_{uc} that is reachable from (x_{H0}, x_{G0}). For $(x_H, x_G) \in X_R^{\uparrow}$, Definition 2.8 implies:

   (a) $\forall \sigma_{uc} \in \Sigma_{uc}$ if $x_H \xrightarrow{\sigma_{uc}} x'_H$, then $\exists x'_G$ such that $x_G \xrightarrow{\sigma_{uc}} x'_G$ and $(x'_H, x'_G) \in X_R^{\uparrow}$

   (b) $\forall \sigma_{uc} \in \Sigma_{uc}$ if $x_G \xrightarrow{\sigma_{uc}} x'_G$, then $\exists x'_H$ such that $x_H \xrightarrow{\sigma_{uc}} x'_H$ and $(x'_H, x'_G) \in X_R^{\uparrow}$.

   Now, let $(\tilde{x}_H, \tilde{x}_G) \in S^{\uparrow}$. By language inclusion:

$$\forall \sigma_{uc} \in \Sigma_{uc} \text{ if } \tilde{x}_H \xrightarrow{\sigma_{uc}} \tilde{x}'_H, \text{ then } \exists \tilde{x}'_G \text{ such that } \tilde{x}_G \xrightarrow{\sigma_{uc}} \tilde{x}'_G \text{ and } (\tilde{x}'_H, \tilde{x}'_G) \in S^{\uparrow}.$$

   By controllability:

$$\forall \sigma_{uc} \in \Sigma_{uc} \text{ if } \tilde{x}_G \xrightarrow{\sigma_{uc}} \tilde{x}'_G, \text{ then } \exists \tilde{x}'_H \text{ such that } \tilde{x}_H \xrightarrow{\sigma_{uc}} \tilde{x}'_H \text{ and } (\tilde{x}'_H, \tilde{x}'_G) \in S^{\uparrow}.$$

   $S^{\uparrow}$ is a bisimulation relation with respect to Σ_{uc} and is as large as possible (by the supremality of $\mathcal{L}(H)^{\uparrow}$), further $S^{\uparrow}$ only contains state pairs reachable from (x_{H0}, x_{G0}). $X_R^{\uparrow}$ is the greatest bisimulation with respect to Σ_{uc} reachable from (x_{H0}, x_{G0}); hence, $S^{\uparrow} = X_R^{\uparrow}$, and by the construction of $X_R^{\uparrow}$, (x_H, x_G) state pairs can only be members of $S^{\uparrow}$ if they are reachable from (x_{H0}, x_{G0}) by sequences of state transitions that remain in Φ_{uc}. As a consequence, $\mathcal{L}(H)^{\uparrow} = \mathcal{L}(R)$. ■

The greatest bisimulation relation between H and G with respect to Σ_{uc}, Φ_{uc}, can be found by partition refinement in time bounded by $O(|\Sigma_{uc}|(n_H + n_G)\log(n_H + n_G))$ (Fernandez, 1990), where n_H and n_G are the number of states in H and G respectively. Perhaps limiting the importance of this result is that a reachability test is still required to eliminate extraneous unreachable states, i.e., forming the greatest bisimulation relation in a running time of $O(|\Sigma_{uc}|(n_H + n_G)\log(n_H + n_G))$ does not allow $\mathcal{L}(H)^\uparrow$ to be determined in '$m\log(n)$', too. This issue is currently being investigated; however, for many applications (such as constructing on-line supervisors) it is often the case that reachability is not of concern.

4. Solution to the Supervisory Control Problem using Bisimulation

4.1. Off-Line Solution to the BSCP

Let a DES and desired behavior be modeled as in Definition 3.1 by G and H, respectively, where $\mathcal{L}(H) \subseteq \mathcal{L}(G)$ and $\mathcal{L}(H)^\uparrow = \mathcal{L}([H \times G]^\uparrow)$. Let every state be marked in G and H, and let the set of uncontrollable events be Σ_{uc}.

Presented here is an algorithm, BISIM-BSCP, that uses bisimulation to find a DFSA that generates $\mathcal{L}(H)^\uparrow$.

Off-Line Algorithm: BISIM-BSCP

Step 1. Find the coarsest stable relational partition, Π, of G and H with respect to the set of uncontrollable events, Σ_{uc}, using the algorithm in (Fernandez, 1990). Denote by Φ_Π the greatest bisimulation relation of H and G (with respect to Σ_{uc}) induced by Π.

Step 2. If (x_{H0}, x_{G0}) is not in the equivalence relation represented by Π, then let R be the empty automaton. Otherwise, let $R = (X_R, \Sigma_R, \delta_R, act_R, x_{R0})$ where

$$
\begin{aligned}
X_R &= \{x_R \in X_H \times X_G | x_H \in X_H, x_G \in X_G, x_R = (x_H, x_G) \in \Phi_\Pi\} \\
x_{R0} &= (x_{H0}, x_{G0}) \\
\forall \sigma \in \Sigma &: \\
\delta_R((x_H, x_G), \sigma) &= \begin{cases} (\delta_H(x_H, \sigma), \delta_G(x_G, \sigma)) & \text{if } (\delta_H(x_H, \sigma), \delta_G(x_G, \sigma)) \in X_R \\ \text{undefined} & \text{otherwise} \end{cases} \\
act_R((x_H, x_G)) &= \{\sigma \in \Sigma | \delta_R((x_H, x_G), \sigma) \text{ is defined}\}.
\end{aligned}
$$

Step 3. Let $R^\uparrow$ be the accessible submachine of R.

THEOREM 4.1 *The automaton $R^\uparrow$ generated by following the algorithm* BISIM-BSCP *generates the supremal controllable sublanguage of $\mathcal{L}(H)$ with respect to $\mathcal{L}(G)$ and Σ_{uc}. If (x_{H0}, x_{G0}) is not in the bisimulation relation induced by the partition Π, then R is the empty automaton and $\mathcal{L}(H)^\uparrow = \emptyset$. Otherwise, $\mathcal{L}(R^\uparrow) = \mathcal{L}(H)^\uparrow$.*

Proof 4.1: Follows from the results of Section 3. ■

4.2. Solving the Nonblocking Version of the BSCP

The non-blocking version of the Basic Supervisory Control Problem can also be solved using bisimulation relations by iterating on Steps 1-3 of the above algorithm. More specifically, let a DES be modeled by the DFSA $G = (X_G, \Sigma_G, \delta_G, act_G, x_{G0}, X_{Gm})$, and let a desired behavior be modeled by the DFSA $H = (X_H, \Sigma_H, \delta_H, act_H, x_{H0}, X_{Hm})$. Considered here is the case when $X_{Gm} \neq X_G$ and $X_{Hm} \neq X_H$. Assume that $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G)$ and $\mathcal{L}(H) \subseteq \mathcal{L}(G)$. Let $\mathcal{L}_m(H)^\uparrow = \mathcal{L}_m([H \times G]^\uparrow)$, and let the set of uncontrollable events be Σ_{uc}.

Off-Line Non-Blocking Algorithm: BISIM-BSCP-NB

Step 0. Initialize such that

$$\begin{aligned} R^{0\uparrow} &= \text{the empty automaton,} \\ H^0 &= H, \\ \Pi^0 &= \{X_H \cup X_G\} \\ i &= 0. \end{aligned}$$

Step 1. $i = i + 1$. Let $H^i = (X_H^i, \Sigma_H, \delta_H^i, act_H, x_{H0}^i, X_{Hm}^i)$, where

$$\begin{aligned} X_H^i &= \{x_H \in X_H | \exists x_G \in X_G \text{ such that } (x_H, x_G) \in \Phi_{\Pi^{i-1}}\} \\ \delta_H^i(x_H, \sigma) &= \begin{cases} \delta_H(x_H, \sigma) & \text{if } \delta_H(x_H, \sigma) \in X_H^i \\ \text{undefined} & \text{otherwise} \end{cases} \\ X_{Hm}^i &= X_{Hm} \cap X_H^i. \end{aligned}$$

Find the coarsest stable relational partition, Π^i, of G and H^i with respect to the set of uncontrollable events, Σ_{uc}, using the algorithm in (Fernandez, 1990).

Step 2. If (x_{H0}, x_{G0}) is not in the greatest bisimulation relation of H^i and G (with respect to Σ_{uc}) induced by Π^i, then let R^i be the empty automaton. Otherwise, let $R^i = (X_R^i, \Sigma_R, \delta_R^i, act_R^i, x_{R0}, X_{Rm}^i)$ where

$$\begin{aligned} X_R^i &= \{x_R \in X_H^i \times X_G | x_H \in X_H^i, x_G \in X_G, x_R = (x_H, x_G) \in \Phi_{\Pi^i}\} \\ X_{Rm}^i &= \{x_R \in X_H^i \times X_G | x_H \in X_{Hm}^i, x_G \in X_{Gm}, x_R = (x_H, x_G) \in \Phi_{\Pi^i}\} \\ x_{R0} &= (x_{H0}, x_{G0}) \\ \forall \sigma \in \Sigma &: \\ \delta_R^i((x_H, x_G), \sigma) &= \begin{cases} (\delta_H^i(x_H, \sigma), \delta_G(x_G, \sigma)) & \text{if } (\delta_H^i(x_H, \sigma), \delta_G(x_G, \sigma)) \in X_R^i \\ \text{undefined} & \text{otherwise} \end{cases} \\ act_R^i((x_H, x_G)) &= \{\sigma \in \Sigma | \delta_R^i((x_H, x_G), \sigma) \text{ is defined}\}. \end{aligned}$$

Step 3. Let $R^{i\uparrow} = \text{trim}(R^i)$. If $R^{i\uparrow} = R^{i-1\uparrow}$, then stop; otherwise go to **Step 1**.

THEOREM 4.2 *The automaton $R^{i\uparrow}$ generated at convergence following the algorithm* BISIM-BSCP-NB *generates the supremal controllable sublanguage of $\mathcal{L}_m(H)$ with respect to $\mathcal{L}(G)$ and Σ_{uc}. If $R^{i\uparrow}$ is the empty automaton, then $\mathcal{L}_m(H)^\uparrow = \emptyset$; otherwise $\mathcal{L}_m(R^{i\uparrow}) = \mathcal{L}_m(H)^\uparrow$.*

Figure 4. H, Automaton representing desired language.

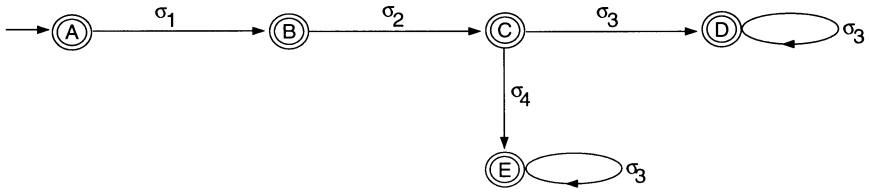

Figure 5. G, Automaton representing plant.

Proof 4.2: Follows from the results of Section 3 and the standard algorithm (Wonham and Ramadge, 1987) for solving the BSCP-NB. ■

4.3. BSCP Solution Example

This subsection demonstrates the off-line solution of the Basic Supervisory Control Problem using the greatest bisimulation with respect to the set of uncontrollable events. An automaton that represents a desired language is shown in Figure 4. The plant automaton is shown in Figure 5. Assume $\Sigma_{uc} = \{\sigma_4\}$.

To find the coarsest stable relational partition with respect to Σ_{uc}, we begin by forming a partition, Π_0, of the union of all the states in H and G:

$$\Pi_0 = \{I\} = \{\{A, B, C, D, E, a, b, c, d\}\}$$

and determine the preimage set of block I due to the uncontrollable event to be:

$$E^{-1}_{[\sigma_4]}(\{A, B, C, D, E, a, b, c, d\}) = \{C\}.$$

Refining the initial partition with respect to $E^{-1}_{[\sigma_4]}(I)$ yields the partition:

$$\Pi_1 = \{II, III\} = \{\{A, B, D, E, a, b, c, d\}, \{C\}\}$$

which is stable with respect to both $E^{-1}_{[\sigma_4]}(II)$ and $E^{-1}_{[\sigma_4]}(III)$ causing the partition refinement algorithm to terminate. $R^{\uparrow}$, shown in Figure 6, is the reachable part of the product machine over possible state pairs in Φ_{Π_1} (cf. Step 3 of BISIM-BSCP) where

$$\begin{aligned}\Phi_{\Pi_1} \;=\; & \{(A, a), (A, b), (A, c), (A, d), (B, a), (B, b), (B, c), (B, d),\\ & (D, a), (D, b), (D, c), (D, d), (E, a), (E, b), (E, c), (E, d)\}.\end{aligned}$$

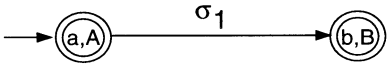

Figure 6. Solution to BSCP.

4.4. Discussion of Computational Complexity

Off-Line

Inspection of the off-line solution of Section 4.2 to the Basic Supervisory Control Problem reveals that finding a partition that constrains the state-pairings allowed for the standard realization (Cassandras *et al*., 1995) of the supervisor can be done in $O(|\Sigma_{uc}|(n_H+n_G)\log(n_H+n_G))$. *Actually constructing* $R^{\uparrow}$, the standard realization of the supervisor, by forming the state-pairs still maintains the worst-case complexity, $O(|\Sigma|n_H n_G)$, as previous efficient methods for computing the supremal controllable sublanguage when the marked language is prefix-closed.

As previous discussed, when the marked language is not prefix-closed computing controllable sublanguage and trimming must be iterated upon. Finding the partition representing the greatest bisimulation relation with respect to Σ_{uc} in $O(m\log(n))$ allows efficient determination of states that violate the controllability condition. This additional information is useful in Step 1 of each iteration of BISIM-BSCP-NB and reduces the expected complexity as compared to the equivalent step in the standard algorithm which constructs the product automaton *first* with worst-case complexity of $O(mn)$.

The primary advantage of using partition refinement is that information is obtained about which states violate controllability *before* the product automaton is formed. The improved algorithm may be more practical for finding supervisory controllers of larger systems, as discussed below, where constructing the entire product automaton is not desirable.

On-Line

If one has a small amount of computing power available for on-line computations, then the reachability test of Step 3 of BISIM-BSCP can be avoided if the supervisor is generated on-line, and Step 2 of BISIM-BSCP only requires a $O(|\Sigma_c|)$ complexity "1-step-lookahead" search over the set of controllable events, Σ_c, at each current state to ensure that the bisimulation relation is not violated. The storage requirements for such an on-line supervisor is $O(|\Sigma|(n_H+n_G))$, for only the partition associated with the greatest bisimulation relation with respect to Σ_{uc}, the plant model, G, and the specification model, H, need to be stored (and *not* the product).

Specifically, the coarsest relational partition of $X_H \cup X_G$ with respect to Σ_{uc}, Π, is determined off-line and is one of the components stored by the on-line supervisor. The automata models of H and G are also stored by the on-line supervisor. During operation, the on-line supervisor observes an event and determines the current state in the plant model

and the specification model from their respective transition functions. The supervisor then uses the model transition functions and performs a "1-step-lookahead" search to determine which state pairs are immediately reachable by controllable events; this operation is done in $O(|\Sigma_c|)$ time. The supervisor then checks the list of state pairs to determine which controllable events need to be disabled to prevent the bisimulation relation from being violated. More precisely, for each pair of possible next states (x'_H, x'_G), if x'_H and x'_G are not in the same block of Π, then the event leading to (x'_H, x'_G) is disabled. Checking a state pair against Π requires $O(1)$ time. In summary, off-line calculations for the on-line supervisor require $O(|\Sigma_{uc}|(n_H+n_G)\log(n_H+n_G))$ time, and the on-line calculations are of complexity $O(|\Sigma_c|)$ at each state. On-line storage is $O(|\Sigma|(n_H+n_G))$.

5. Input/Output Finite-State Machine Model Matching

The problem of strong model matching for finite-state machines (FSMs), as posed by (DiBenedetto *et al*., 1994, 1995, 1996), consists of finding a controller for a given open loop system that results in a desired closed loop *behavior* (cf. precise statement in Section 5.1). The resulting controller maps command signals in some command set *V* to control signals in some control set *U*. It is claimed in (DiBenedetto *et al*., 1995, 1996) that the supervisory control problem in its basic form can be posed as a special case of model matching for finite-state machines. This claim suggests I/O FSM model matching may be more general than the feedback loop of supervisory control in the Ramadge-Wonham paradigm (Ramadge and Wonham, 1987, 1989). This section introduces strong I/O FSM model matching. In Section 7, the above claim will be formally investigated. The remainder of this section is organized as follows. Subsection 5.1 provides introductory material covering I/O finite-state machines, FSM behavior and how behavioral equivalence relates to bisimulation. The decision problems associated with strong I/O FSM model matching are presented in Subsection 5.2, and Subsection 5.3 presents some behavioral inclusion problems associated with strong I/O FSM model matching.

5.1. Input/Output FSMs, Behavioral Equivalence, and Bisimulation

This section deals primarily with basic definitions for FSMs and operations between FSMs as given in (DiBenedetto *et al*., 1994, 1995, 1996). As in (DiBenedetto *et al*., 1995, 1996), Greek letters are used for functions or relations, capital English letters represent sets, and lower case represent elements. The numbers associated with definitions and theorems correspond to section numbers used in this paper and are not necessarily those given in (DiBenedetto *et al*., 1994, 1995, 1996) unless otherwise stated.

DEFINITION 5.1 *(DiBenedetto et al., 1995) An* **input/output finite-state machine** *(I/O FSM) is a 6-tuple $F=(I, O, S, \lambda, \gamma, r)$ with I the input alphabet, O the output alphabet, S the set of states, $\lambda : I \times S \to 2^S$ the next-state function, $\gamma : I \times S \times S \to 2^O$ the output function, and $r \in S$ the initial state.*

The next-state function for sequences of inputs can be defined recursively. Let $I^k = I \times \cdots \times I$ (k times) where k is a finite integer, $k \geq 1$. The next-state function for a sequence of inputs, $\lambda^k : I^k \times S \to 2^S$ is defined by

$$\begin{aligned} \lambda^1(i_0, s) &= \lambda(i_0, s) \\ \lambda^k(i_0 i_1 \ldots i_{k-1}, s) &= \bigcup_{\tilde{s} \in \lambda^1(i_0, s)} \lambda^{k-1}(i_1 i_2 \ldots i_{k-1}, \tilde{s}), \end{aligned}$$

and an output sequence associated with an input sequence of length k is defined by

$$\gamma^k(i_0 \ldots i_{k-1}, r) = \{(o_0 \ldots o_l \ldots o_{k-1}) | o_l \in \gamma(i_l, s_l, s_{l+1}) \wedge s_{l+1} \in \lambda(i_l, s_l), l = 0, \ldots, k-1; s_0 = r\}.$$

From the above description of input/output sequences, it is evident that the *behavior* of a FSM in the framework of (DiBenedetto *et al.*, 1994, 1995) is identified with its set of *extended traces* (Arnold, 1994) of the form

$$i_{j_0} O_{j_0} i_{j_1} O_{j_1} \ldots i_{j_{k-1}} O_{j_{k-1}},$$

where $i_{j_*} \in I$ and $O_{j_*} \subseteq O$, $k \geq 1$.

In the sequel, we drop the qualifier I/O for FSMs whenever it will be clear from the context. It is common to label a transition due to input i that generates output o with the label i/o. If λ and γ always map to singletons, i.e., we can view them as functions $\lambda : I \times S \to S$ and $\gamma : I \times S \times S \to O$, then the FSM is called a deterministic finite-state machine (DFSM). Otherwise, the FSM is called a nondeterministic finite-state machine (NDFSM). If there are multiple outputs due to the same input, i, between two states, then this will be represented by multiple transitions with labels i/o_1, i/o_2, etc. Let $\mathcal{I}(s)$ denote the set of applicable inputs to the FSM at state s. Similarly, let $\mathcal{I}^k(s)$ denote the set of applicable input sequences of length k at state s. According to the semantics given in (DiBenedetto *et al.*, 1995), if an input $i \in I$ is not applicable at a state s, then $\lambda(i, s) = \theta$ where θ is a special state called the *dead state*. For every input, the next state relation for the dead state is the dead state. If there are no outputs for a given transition, then $\gamma(i, s, s') = \mu$ where μ is called the *silent output*. All transitions leading to the dead state have the silent output. Hereafter all FSMs will be assumed to have a dead state and every transition defined at every state; the dead state is always identifiable among the elements of S (we will call it θ), and $\mathcal{I}(s)$ is implicitly defined by the knowledge of θ.

To compare FSMs in (DiBenedetto *et al.*, 1994, 1995, 1996) the notion of FSM behavioral equivalence is introduced:

DEFINITION 5.2 *(DiBenedetto et al., 1995)* [10] *Given DFSM $F_1 = (I, O, S_1, \lambda_1, \gamma_1, r_1)$ and NDFSM $F_2 = (I, O, S_2, \lambda_2, \gamma_2, r_2)$, the behavior of F_1 is equivalent to the behavior of F_2, denoted $F_1 =_{\mathcal{B}} F_2$, if*

$$\begin{aligned} &\forall k \geq 1, \forall i_0 i_1 \ldots i_{k-1} \in I^k, \\ &\gamma_1^k(i_0 \ldots i_{k-1}, r_1) = \gamma_2^k(i_0 \ldots i_{k-1}, r_2). \end{aligned}$$

In words, equivalence of FSMs is equivalence of their extended trace sets. Because one of the FSMs must be a DFSM, behavioral equivalence is the same as bisimulation, i.e.,

behavioral equivalence indicates that F_1 and F_2 are bisimilar with respect to the input and output alphabets. To formalize this concept, we present the following theorem.

THEOREM 5.1 *Let F_1 and F_2 be as in Definition 5.2, and let their respective state spaces S_1 and S_2 contain only reachable states. Then $F_1 =_{\mathcal{B}} F_2$ iff F_1 is bisimilar to F_2 with respect to $I \times O$.*

Proof 5.1: Note that marking is not an issue here since F_1 and F_2 are not defined to have marked states; hence, the third condition in the definition of bisimulation is not relevant.

If : Let F_1 be bisimilar to F_2 with respect to $I \times O$, then the normal forms of F_1 and F_2 are isomorphic (and deterministic because F_1 is deterministic), so clearly $F_1 =_{\mathcal{B}} F_2$.

Only if : Let $F_1 =_{\mathcal{B}} F_2$ and construct a binary relation $\mathcal{R}$ as follows.
(1) $(r_1, r_2) \in \mathcal{R}$
(2) $\forall k \geq 1, \forall i_0 i_1 \ldots i_{k-1} \in I^k, \forall r_2' \in \lambda_2^k(i_0 \ldots i_{k-1}, r_2), \quad (\lambda_1^k(i_0 \ldots i_{k-1}, r_1), r_2') \in \mathcal{R}$
By Definition 5.2 and construction of $\mathcal{R}$, for all $s_1 \in S_1$ reachable by sequence $i_0 i_1 \ldots i_j$ there exists a $s_2 \in S_2$ such that $(s_1, s_2) \in \mathcal{R}$. If $(s_1, s_2) \in \mathcal{R}$ and $s_1 \xrightarrow{i/o} s_1'$, then the output o is unique due to the determinism of F_1. There exists s_2' such that $s_2 \xrightarrow{i/o^*} s_2'$ and $o^* = o$ by Definition 5.2 and the fact that o is unique. Because s_1' and s_2' are reachable with the same input sequence, $(s_1', s_2') \in \mathcal{R}$. Similarly, for all $s_2 \in S_2$ reachable by sequence $i_0 i_1 \ldots i_j$ there exists a unique $s_1 \in S_1$ such that $(s_1, s_2) \in \mathcal{R}$. If $(s_1, s_2) \in \mathcal{R}$ and $s_2 \xrightarrow{i/o} s_2'$, then there exists s_1' such that $s_1 \xrightarrow{i/o^*} s_1'$, $o^* = o$ and o is unique by Definition 5.2 and the fact that o^* is unique. s_1' and s_2' are reachable by the same input sequence, so $(s_1', s_2') \in \mathcal{R}$. Thus, $\mathcal{R}$ is a bisimulation relation and F_1 is bisimilar to F_2 with respect to $I \times O$. ■

Evidently, if a NDFSM is behaviorally equivalent to a DFSM, then the normal form of the NDFSM is deterministic. It will be said that F_1 and F_2 are equivalent if $F_1 =_{\mathcal{B}} F_2$. Examining the definition of behavioral equivalence reveals that it is defined in terms of at least one of the FSM being deterministic, and it is this requirement that allows the statement of Theorem 5.1. In general, equivalence of extended trace sets does not imply bisimulation.

The interaction of two I/O FSMs (a controller and a plant) is modeled by the I/O FSM composition operation.

DEFINITION 5.3 *(DiBenedetto et al., 1995)*[11] *Given DFSM M_1 and possibly NDFSM M_2 where*

$$M_1 = (U, Y, S_1, \lambda_1, \gamma_1, r_1)$$

$$M_2 = ((V, Y), U, S_2, \lambda_2, \gamma_2, r_2),$$

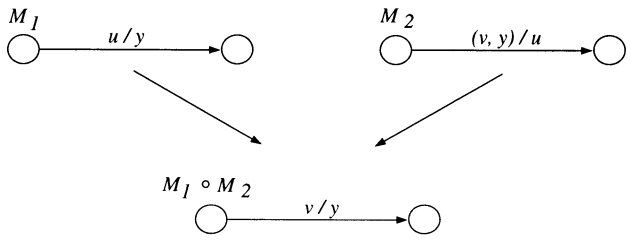

Figure 7. Example of composition.

the **composition** *of M_1 and M_2 is the FSM $M_1 \circ M_2$ denoted $\hat{M} = (V, Y, \hat{S}, \hat{\lambda}, \hat{\gamma}, \hat{r})$ defined as:*

$$
\begin{aligned}
\hat{S} &= S_1 \times S_2 \\
\hat{r} &= (r_1, r_2) \\
\hat{\lambda}(v, (s_1, s_2)) &= \{(s_1', s_2') | \exists u \in U, \exists y \in Y, \\
&\quad s_1' = \lambda_1(u, s_1) \wedge \\
&\quad s_2' \in \lambda_2((v, y), s_2) \wedge \\
&\quad u \in \gamma_2((v, y), s_2, s_2') \wedge \\
&\quad y = \gamma_1(u, s_1, s_1')\} \\
\hat{\gamma}(v, (s_1, s_2), (s_1', s_2')) &= \{y \in Y | \exists u \in U, \\
&\quad s_1' = \lambda_1(u, s_1) \wedge \\
&\quad s_2' \in \lambda_2((v, y), s_2) \wedge \\
&\quad u \in \gamma_2((v, y), s_2, s_2')\}
\end{aligned}
$$

Figure 7 shows an example composition between two FSMs.

5.2. Strong Model Matching Decision Problems

This section presents problems in strong I/O FSM model matching where *exact* matching of behaviors is desired. The Strong FSM Model Matching Problem (SMMP) is as follows.

SMMP : Given DFSMs $M_1 = (U, Y, S_1, \lambda_1, \gamma_1, r_1)$ and $M = (V, Y, S, \lambda, \gamma, r)$, determine a dynamic state feedback compensator M_2, such that

$$
\begin{aligned}
M_2 &= ((V, Y), U, S_2, \lambda_2, \gamma_2, r_2), \text{ and} \\
M_1 \circ M_2 &=_{\mathcal{B}} M.
\end{aligned}
$$

The major result of [(Dibenedetto *et al.*, 1995), Theorem 3.2] states that the SMMP is solvable if and only if the inverse automaton[12] of M, M^{-1}, is simulated by the inverse automaton

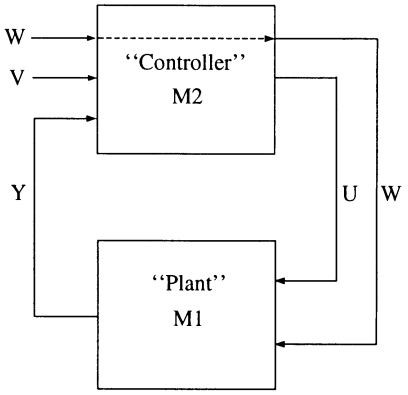

Figure 8. Feedback loop of the SMMP–D.

of M_1, M_1^{-1}. Intuitively, the machine M_2 must map command inputs in V to elements in U which drive the plant FSM M_1. The simulation requirement of the inverse automata of M and M_1 can be viewed as requiring the existence of an M_2 that does not attempt to force the plant, M_1, to produce an output sequence that it cannot.

The problem of strong model matching with disturbance measurement (SMMP–D) is as follows (DiBenedetto *et al.*, 1996).

SMMP–D : Given DFSMs $M_1 = ((W, U), Y, S_1, \lambda_1, \gamma_1, r_1)$ and $M = ((W, V), Y, S, \lambda, \gamma, r)$, determine a dynamic state feedback compensator for M_1, $M_2 = (((W, V), Y), (W, U), S_2, \lambda_2, \gamma_2, r_2)$, with $\gamma_2 = (\gamma_2^w, \gamma_2^u)$ and $\gamma_2^w = ((W, V),Y) \times S_2 \times S_2 \rightarrow W$, $\gamma_2^u = ((W, V), Y) \times S_2 \times S_2 \rightarrow U$ such that γ_2^w is the canonical projection and

1. $M_1 \circ M_2 =_{\mathcal{B}} M$.
2. $\forall (s_1, s_2) \in \hat{S}, \mathcal{W}(s_1, s_2) = \mathcal{W}(s_1)$ where $\mathcal{W}(s)$ is the set of disturbances applicable at state s.

The canonical projection is simply the identity mapping of elements in W from input to output, i.e., if w appears in the input set, $((W, V), Y)$, for a particular transition, then w must appear in the output set, (W, U), of that transition. With the introduction of measureable disturbances, the feedback loop of the SMMP–D is described as that shown in Figure 8 (see also [(DiBenedetto *et al.*, 1996), Figure 9]).

No formal definition of I/O FSM composition is given in (DiBenedetto *et al.*, 1996) for the case of measurable disturbances; however, from its description and for the purposes here, Figure 9 is taken as representative of FSM composition with disturbances. We assume that only one input can occur at a time, i.e., a disturbance cannot occur while a command is given and vice versa.

New machines are defined in (DiBenedetto *et al.*, 1995) for the case of measurable

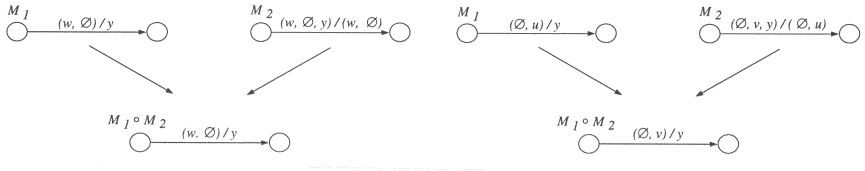

Figure 9. Generalization of FSM composition when disturbances are present.

disturbances. Machines $\tilde{M}$ and $\tilde{M}_1$ are modifications of M and M_1 where the disturbances are present at the input and output of both machines. Theorem 4.1 of (DiBenedetto *et al*., 1995) states that the SMMP-D is solvable if and only if the inverse automaton $\tilde{M}^{-1}$ is simulated by the inverse automaton $\tilde{M}_1^{-1}$. The intuition for this result about the SMMP-D is the same as for the SMMP with the constraint that M_2 cannot alter disturbances.

Both the SMMP and the SMMP–D are *decision* problems. Once a reference model is created, all that need be checked is whether the reference model is simulated by the plant, i.e., the desired behavior is feasible or not. This checking is performed by creating the greatest simulation relation between the reference and the plant models which, fortunately, also provides the correct state matching to perform the modified product. Thus, forming the simulation relation decides and produces the solution to the problem.

If a solution does not exist, i.e., exact matching cannot occur, there is no indication from the SMMP as to what sub-behavior *could* be matched. In order for the Basic Supervisory Control Problem to be solved in the I/O FSM Model Matching framework, there needs to be some mechanism for determining maximal or supremal solutions, that is, there must be some way to cast model matching decision problems as optimization problems.

5.3. Strong Model Matching Inclusion Problems

As discussed in the previous section, the case of *exact* model matching is not enough to allow supervisory control problems to be solved using the formalisms discussed in (DiBenedetto *et al*., 1994, 1995, 1996). Additionally, there must exist some methodology of dealing with matching some maximal set of desired behaviors. This problem is addressed in (DiBenedetto *et al*., 1995, 1996) as matching a set of behaviors *contained* in a reference model.

DEFINITION 5.4 (DiBenedetto *et al*., 1996) *Given NDFSMs* $F_1 = (I, O, S_1, \lambda_1, \gamma_1, r_1)$ *and* $F_2 = (I, O, S_2, \lambda_2, \gamma_2, r_2)$, *the behavior of* F_1 *is* **contained** *in the behavior of* F_2, *denoted* $F_1 \subseteq_{\mathcal{B}} F_2$, *if*

$$\forall k \geq 1,\ \forall i_0 i_1 \ldots i_{k-1} \in \mathcal{I}^k(r_1),$$

$$\gamma_1^k(i_0 \ldots i_{k-1}, r_1) \subseteq \gamma_2^k(i_0 \ldots i_{k-1}, r_2).$$

This definition implies that the set of all input-output sequences of F_1 such that the input sequence does not take F_1 to the dead state is a subset of the set of input-output sequences of F_2.

The problem of strong FSM model matching with a set of sub-behaviors contained in a reference models is given in (DiBenedetto *et al.*, 1995) as:

SMMP–SB : Given DFSM $M_1 = (U, Y, S_1, \lambda_1, \gamma_1, r_1)$ and NDFSM $M = (V, Y, S, \lambda, \gamma, r)$, determine a dynamic state feedback compensator for M_1, $M_2 = ((V, Y), U, S_2, \lambda_2, \gamma_2, r_2)$ such that $M_1 \circ M_2 =_{\mathcal{B}} \hat{M}$ where $\hat{M} = (V, Y, \hat{S}, \hat{\lambda}, \hat{\gamma}, \hat{r})$ is a nontrivial DFSM and $\hat{M} \subseteq_{\mathcal{B}} M$.

For the sake of notational simplicity (DiBenedetto *et al.*, 1995), the following notation is used for a NDFSM $F = (I, O, S, \lambda, \gamma, r)$: $F = (I, O, S, \Delta, r)$ where $\Delta = \{(i, s, s', o) \in I \times S \times S \times O | s' \in \lambda(i, s) \wedge o \in \gamma(i, s, s')\}$.

Given $M = (V, Y, S, \Delta, r)$ and $M_1 = (U, Y, S_1, \Delta_1, r_1)$, let $\overline{M} = (V, Y, \overline{S}, \overline{\Delta}, \overline{r})$ be defined as follows[13]:

$$
\begin{aligned}
\overline{r} &= (r_1, r) \\
\overline{S} &= \{\overline{s} \in S_1 \times S | \overline{s} \text{ is reachable from } \overline{r}\} \\
\overline{s} &= (s_1, s) \\
\overline{s}' &= (s_1', s') \\
(v, \overline{s}, \overline{s}', y) \in \overline{\Delta} \quad &\text{if} \quad \exists u \in U \text{ such that } (u, s_1, s_1', y) \in \Delta_1 \wedge \\
&\qquad (v, s, s', y) \in \Delta \\
(v, \overline{s}, \overline{\theta}, y) \in \overline{\Delta} \quad &\text{otherwise.}
\end{aligned}
$$

To solve the SMMP-SB problem, any particular deterministic behavior in $\overline{M}$ can be chosen and a controller constructed by looking at the V, U and Y information at the corresponding state-pairs of $\overline{M}$. (DiBenedetto *et al.*, 1995) state that the language associated with $\overline{M}$ is the supremal controllable language in the supervisory control framework; however, since there are no uncontrollable events or disturbances associated with $\overline{M}$, this fact is immediate and it cannot be used as a basis to state that the Basic Supervisory Control Problem is a special case of strong I/O FSM model matching.

Generalizing their earlier results, DiBenedetto *et al.* recently produced (DiBenedetto *et al.*, 1996) which gives results for the Strong I/O FSM Model Matching with measurable disturbances to any set of deterministic behaviors contained in a reference model. This problem is abbreviated here as SMMP-D-SB and the problem statement is given as follows.

SMMP-D-SB : Given DFSM $M_1 = ((W, U), Y, S_1, \lambda_1, \gamma_1, r_1)$ and NDFSM $M = ((W, V), Y, S, \lambda, \gamma, r)$, determine a dynamic state feedback compensator for M_1, $M_2 = ((W, V, Y), (W, U), S_2, \lambda_2, \gamma_2, r_2)$ such that

1. $M_1 \circ M_2 =_{\mathcal{B}} \hat{M}$ where $\hat{M} = ((W, V), Y, \hat{S}, \hat{\lambda}, \hat{\gamma}, \hat{r})$ is a nontrivial DFSM and $\hat{M} \subseteq_{\mathcal{B}} M$.

2. $\forall(s_1, s_2) \in \hat{S}, \mathcal{W}(s_1, s_2) = \mathcal{W}(s_1)$ where $\mathcal{W}(s)$ is the set of disturbances applicable at state s.

In addition to the above problem, DiBenedetto *et al*. derive a characterization of the set of maximal deterministic behaviors that can be matched given the constraint that the reference model M is an output-deterministic FSM (ODFSM). An ODFSM has the property that for any state and any input, for each unique output generated at a state, there is a unique next state.

A compensator, M_2, is said to solve the maximal behavior problem (SMMP–D–MB) if, in addition to the SMMP–D–SB requirements, the resulting closed loop behavior, $\hat{M}$, is as large as possible, that is:

SMMP-D-MB : SMMP-D-SB +

3. $\hat{M}$ is as large as possible, i.e., if there exists any other compensator M_2^* such that $M_1 \circ M_2^* =_{\mathcal{B}} \hat{M}^* \subseteq_{\mathcal{B}} M$, $\hat{M} \subseteq_{\mathcal{B}} \hat{M}^*$ and $\forall(s_1, s_2^*) \in \hat{S}^*, \mathcal{W}(s_1, s_2^*) = \mathcal{W}(s_1)$, then $\hat{M} =_{\mathcal{B}} \hat{M}^*$.

The following sections present an alternative method for solving, when M_1 and M are DFSMs, the Strong I/O FSM Model Matching with Measurable Disturbances to a Maximal Set of Allowable Behaviors in a Reference Model Problem. The problem is solved by being posed in the framework of SCT as an instance of BSCP–NB with multiple marking classes. Solving the SMMP–D–MB as the BSCP–NB utilizes two key relationships that we derive:

1. behavioral inclusion is related to marked language inclusion, and

2. languages associated with matchable behaviors in the presence of disturbances must be controllable.

Note that we require the reference model, M, to be deterministic. The procedure presented also assumes that the "plant" M_1 is a DFSM which is consistent with the algorithms presented in (DiBenedetto *et al*., 1996). Prior to establishing a formal link between SMMP and SCT, a method is presented for converting discrete-event models from one framework to another.

6. Discrete-Event Model Conversions

In order to investigate the relationship between I/O FSM model matching and Supervisory Control Theory, it is necessary to take models in one framework and modify them to fit into another framework. This section discusses a means to convert models in the SCT framework to I/O FSM models in the Strong I/O FSM Model Matching framework, and vice-versa.

6.1. Converting SCT Automata Models to I/O FSM Models

Translating a possibly nondeterministic automaton from the supervisory control framework to a I/O FSM of the model matching framework utilizes the "classical" method to convert a Moore type machine to a Mealy type machine. The conversion procedure involves moving "output labels" from states to state transitions. Further, uncontrollable events for automata in the SCT framework are viewed as disturbances in I/O FSM models and transition labels are constructed to reflect this analogy. Let the event σ take the automaton from state s to state s', i.e.,

$$s \xrightarrow{\sigma} s'.$$

If an output o is associated with state s' then the corresponding transition in the I/O FSM is assigned the transition:

$$s \xrightarrow{\sigma/o} s'.$$

In the supervisory control framework it is often the case that outputs are not assigned to states; thus, there is no state output to "move" to the transition label. In this treatment of the problem the marking class of a state will serve as the output component of the appropriate transition label.

In SCT, the marking of a state generally represents some particular action of interest such as a completed task. The notion of state marking is a special case of the state *parameters* discussed in (Arnold, 1994). Here, as in (Thistle *et al*., 1995), state marking will be generalized to a class of possible state markings. Instead of considering a single set of marked states, X_m, we will consider several sets of marked states associated with each automaton. Denote the set of marking classes of an automaton with the indexed set $\mathcal{X}_m = \{X_{m_1}, X_{m_2}, \ldots, X_{m_k}\}$. Different automata can have different sets of marking classes, but it will be maintained that two marking classes with the same index hold the same "meaning" (e.g. similar indices represent similar output values.) Utilizing many sets of marked states in a single automaton to represent a family of marked languages is more convenient than using many automata, one for each marked language. The markings of a composite state resulting from the product ($\times$) or parallel composition ($||$) of two automata are the intersection of the markings of similar index of the two component states, i.e., a composite state is marked m_i if each component state is marked m_i. As mentioned above, an output can be associated with a state; hence, an output marking class, m_i, is used here to denote the specific output, o_i, of a state. Define an indexed set $O = \{o_1, o_2, \ldots, o_k\}$ such that the index outputs correspond with marking classes of similar index. The function that relates states to their corresponding outputs will be $h : X \to O \cup \{\mu\}$, where

$$h(x) = \begin{cases} o_i & \text{if } x \in X_{m_i} \\ \mu & \text{otherwise.} \end{cases} \tag{3}$$

Each automaton, A_j, can have an associated output function h_j; however, each output function maps to the same co-domain, $O \cup \{\mu\}$.

The set of states and number of transitions remains unchanged when the automaton is "converted" to an I/O FSM. This procedure is described more formally below.

Consider the following procedure to translate an *event* based automaton model to an I/O FSM model. Given a DFSA $A = (X_A, \Sigma_A, \delta_A, act_A, x_{A,0}, \mathcal{X}_{Am})$ let $M = ((W, V), Y, S, \lambda, \gamma, r)$ be the I/O FSM resulting from A. The NULL event, $\emptyset$, is introduced and represents only a "place holder" and is indicative of the absence of an event. Assign the components of M as follows[14]

$$\begin{aligned}
W &= \Sigma_{uc} \cup \{\emptyset\} \\
V &= \Sigma_A \setminus \Sigma_{uc} \cup \{\emptyset\} \\
Y &= O \cup \{\mu\} \\
S &= X \cup \{\theta\} \\
r &= x_{A,0} \\
\lambda &: (W, V) \times S \to 2^S \\
\forall v \in V, \forall w \in W & \\
\lambda((\emptyset, v), s) &= \delta_A(s, v) \text{ if } v \in act_A(s) \setminus \Sigma_{uc} \\
\lambda((w, \emptyset), s) &= \delta_A(s, w) \text{ if } w \in act_A(s) \cap \Sigma_{uc} \\
&= \theta \text{ otherwise.} \\
\gamma &: (W, V) \times S \times S \to 2^Y \\
\gamma((w, v), s, s') &= h(s') \text{ if } s' \in X_A \\
&= \mu \text{ otherwise.}
\end{aligned}$$

Once A is converted to M, denoted by $A \overset{IO}{\longmapsto} M$, each transition in M will have a label of the form $(W, V)/Y$.

6.2. Converting I/O FSM Models to SCT Automaton Models

The conversion of an I/O FSM to an automaton is equivalent to converting a Mealy machine to a Moore machine with the interpretation of state outputs being markings. The procedure described below explicitly uses the "classical" method of converting a Mealy machine to a Moore machine (Hayes, 1993, Kohavi, 1978).

Converting a Mealy machine to a Moore machine is slightly more complicated than renaming transitions as in the Moore-to-Mealy case. Briefly, for each state s' of the I/O FSM with transitions of the form:

$$s \xrightarrow{v/o} s'$$

a new state is introduced into the automaton for each distinct output value o among the transitions entering that state. Thus, if o assumes k distinct values $y_1, y_2, \ldots y_k$ then s' is replaced with k new states $s'_1, s'_2, \ldots s'_k$. Each new state s'_i is assigned the fixed output y_i and supplied with the original transitions. Define an indexed set of marking classes $\mathcal{X}_m = \{X_{m_1}, X_{m_2}, \ldots, X_{m_k}\}$ and assign states as elements of a marking class with index similar to that of the state output. Denote by $M \overset{sct}{\longmapsto} A$ that the Mealy-type machine M converts to the automaton A interpreted as a Moore-type machine. Because it does not add

information to the model, we will assume the dead state of the Mealy-type machine is not kept in the resulting Moore-type automaton. Figure 10 shows the conversion of a Mealy LIFO stack control unit state diagram to a Moore type diagram (Hayes, 1993). The resulting Moore LIFO machine has four marking classes: $\{\emptyset, ERR, LS, RS\}$, one for each distinct output of the Mealy LIFO control unit. As discussed in Section 6.1, the output marking of a particular state s is denoted $h(s)$, e.g., in Figure 10(b) (outputs are in bold) it can be seen that $h(s_1'') = RS$[15]. The initial state is given the output value $\emptyset$ by default. In addition to the output marking classes, m_i, a marking class m_{os} is also defined and holds the meaning "output state". If a state is a member of any output marking class, then it is also assigned to be a member of m_{os}. States that are m_{os}-marked are denoted in figures hereafter by a double circle. Some of the properties of the automaton A represented by the Moore LIFO machine in Figure 10 are as follows:

$$\begin{aligned}
X_A &= \{s_0, s_0', s_0'', s_1, s_1', s_1'', s_2, s_2', s_2''\} \\
\mathcal{X}_m &= \{X_{m_\emptyset}, X_{m_{ERR}}, X_{m_{LS}}, X_{m_{RS}}, X_{m_{os}}\} \\
X_{m_\emptyset} &= \{s_0, s_1, s_2\} \\
X_{m_{ERR}} &= \{s_0'', s_2'\} \\
X_{m_{LS}} &= \{s_0', s_1'\} \\
X_{m_{RS}} &= \{s_1'', s_2''\} \\
X_{m_{os}} &= \{s_0, s_0', s_0'', s_1, s_1', s_1'', s_2, s_2', s_2''\}.
\end{aligned}$$

6.3. Modeling the Strong Model Matching Feedback Loop

To determine the "feasible" or joint behavior between what is required by the specification and what can be performed by the plant, the behavior of the Moore automata must be intersected in some way. For DES modeled as automata, it is common to form the behavioral intersection by composing the specification automaton with the plant automaton.

The finite-state machines used in model matching generally have different input alphabets, i.e., M_1 (the "plant") is defined with input alphabet U, and M (the "specification") is defined with input alphabet V. Because of this, the Moore machine versions of M_1 and M cannot be directly composed using the product composition ($\times$) of SCT. The FSM composition operation ($\circ$) of strong model matching must therefore be modeled to allow the *synchronization* of the Moore machines.

The FSM composition operation allows an output y to occur iff a disturbance w produces y or an input sequence vu produces y. After an output is allowed, the composition operator checks the state reached by the previous sequence to determine if another valid sequence produces an output. Thus, synchronization of all possible sequences is easily modeled as composition with the regular expression $(W + VU)^*$, where W represents all disturbances, V represents all command inputs and U represents all control inputs. Figure 11 shows the automaton *Sync* representing the event ordering effect of the I/O FSM composition

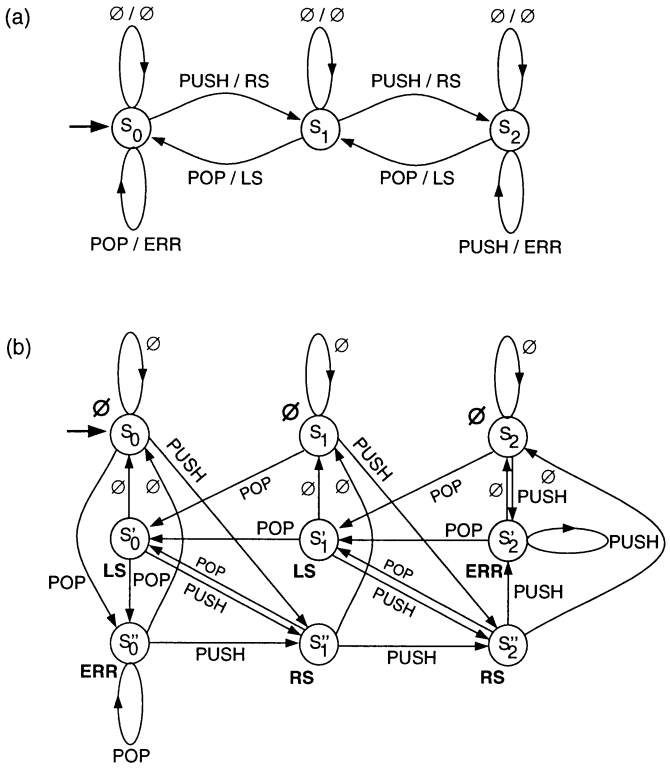

Figure 10. LIFO stack control unit: (a) Mealy state diagram, (b) Moore state diagram.

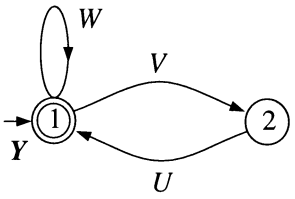

Figure 11. Synchronizer automaton, *Sync*, that models SMMP feedback loop.

operation. The initial state of the composition synchronizer is marked as belonging to all output classes (denoted by Y in the figure).

For example, consider the conversion of M and M_1 to H and G respectively shown in Figure 12. The resulting Moore automata have differing alphabets in that H is defined over $W \cup V$ and G is defined over $W \cup U$. The feasible behavior is formed by synchronizing

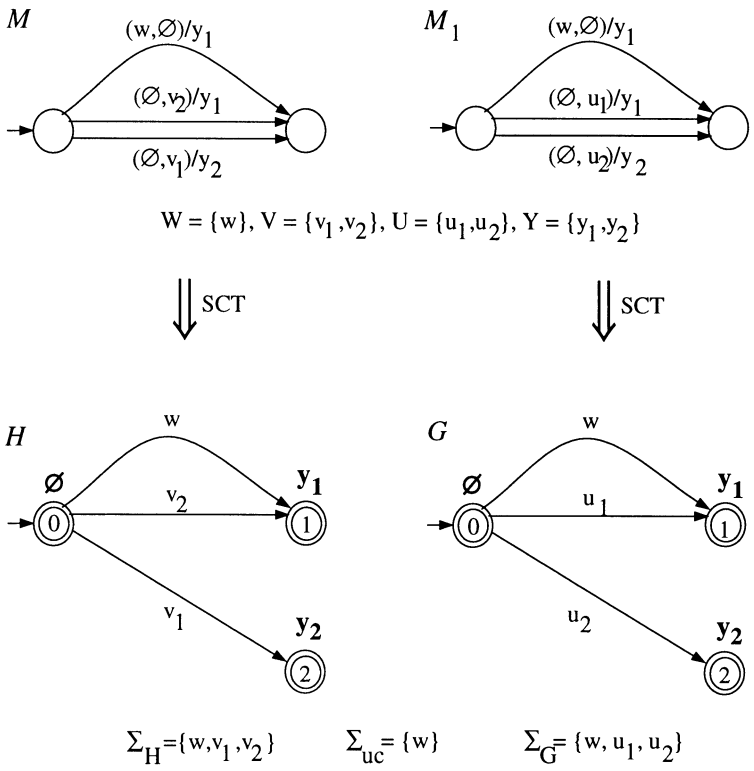

Figure 12. Example machines M_1 and M and their associated Moore automata.

the specification H and plant G with the feedback-loop model *Sync*. Performing this synchronization yields the Moore automata shown in Figure 13.

Examining Figure 13 reveals several interesting artifacts. First, the automaton $\tilde{G}$ has traces which do not lead to output states. This arises due to the fact that after the production of a y_1 or y_2 output, no u input exists to extend the second v sequences to output states. Another interesting artifact is that some m_{os}-marked states in $\tilde{H}$ do not have an associated output marking class. This artifact results from the fact that the output classes of the composite states may not match; hence, during the parallel composition no output marking class can be assigned. These artifacts represent desired behavior that cannot be "matched" by the plant. Future sections will discuss how to procedurally address this issue.

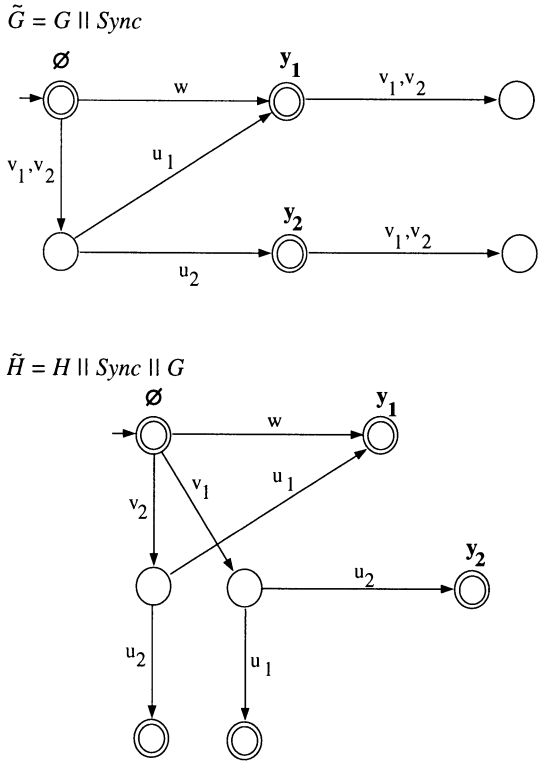

Figure 13. Feasible behaviors formed by synchronization.

7. Formal Comparison of Input/Output FSM Model Matching and Supervisory Control

As stated earlier, it was claimed in (DiBenedetto *et al.*, 1995, 1996) that the supervisory control problem can be posed as a special case of model matching for finite-state machines. This claim is investigated more thoroughly in this section. Specifically, the above conjecture found in (DiBenedetto *et al.*, 1995, 1996) is confirmed. Furthermore, it is shown that finding a maximal solution to the SMMP–D can be solved as an instance of the BSCP–NB for a *family* of marked languages.

The investigation of the relationship between supervisory control and strong FSM model matching in the presence of disturbances will proceed with the following steps:

1. A relationship is derived between behavioral inclusion and inclusion of marked languages (Section 7.1).

2. A relationship is found between behaviors which are matchable in the presence of

disturbances and the controllability of the marked languages associated with those behaviors (Sections 7.2 and 7.3).

3. A method is presented for solving the BSCP as an instance of strong model matching in the presence of measurable disturbances (Section 7.2).

4. A method is presented for solving the SMMP-D-MB for a set of maximal deterministic behaviors as an instance of BSCP–NB for a family of marked languages (Section 7.3).

7.1. Strong Model Matching and Language Properties

The following lemma reveals the relationship between FSMs and the languages of their associated automata.

LEMMA **[Language Inclusion]:** *Let DFSMs F_1 and F_2 be given such that*

$$F_1 = ((W, V), Y, S_1, \lambda_1, \gamma_1, r_1)$$
$$F_2 = ((W, V), Y, S_2, \lambda_2, \gamma_2, r_2),$$

and $F_1 \overset{sct}{\longmapsto} A_1$, $F_2 \overset{sct}{\longmapsto} A_2$. Then $F_1 \subseteq_{\mathcal{B}} F_2$ if and only if for all $j \in \{1, \ldots, |O|\}$ $\mathcal{L}_{m_j}(A_1) \subseteq \mathcal{L}_{m_j}(A_2)$, where $\mathcal{L}_{m_j}$ is the marked language associated with m_j-marked states.

Recall that $Y = O \cup \{\mu\}$ and that similarly indexed elements of Y and O are equivalent.

Proof [Language Inclusion Lemma] : From the definition of behavioral containment, it is known that:

$$F_1 \subseteq_{\mathcal{B}} F_2 \Leftrightarrow \forall k \geq 1, \forall i_0 i_1 \ldots i_{k-1} \in \mathcal{I}^k(r_1)$$
$$\gamma_1^k(i_0 \ldots i_{k-1}, r_1) = \{(y_0 \ldots y_l \ldots y_{k-1}) | y_l \in \gamma_1(i_l, s_l, s_{l+1}) \wedge s_{l+1} \in \lambda_1(i_l, s_l), l = 0, \ldots, k-1; s_0 = r_1\}$$
$$\gamma_2^k(i_0 \ldots i_{k-1}, r_2) = \{(y_0 \ldots y_l \ldots y_{k-1}) | y_l \in \gamma_2(i_l, s_l, s_{l+1}) \wedge s_{l+1} \in \lambda_2(i_l, s_l), l = 0, \ldots, k-1; s_0 = r_2\}$$
$$[\gamma_1^k(i_0 \ldots i_{k-1}, r_1) = \gamma_2^k(i_0 \ldots i_{k-1}, r_2)].$$

We have equality here because F_1 and F_2 are deterministic, so the output sequence sets, γ_i^k, are singleton sets. From the discussion of Mealy-to-Moore model conversion, the following holds by construction:

$$F_1 \subseteq_{\mathcal{B}} F_2 \Leftrightarrow \forall k \geq 1, \forall i_0 i_1 \ldots i_{k-1} \in \bigcup_{j \in \{1, \ldots, |O|\}} \mathcal{L}_{m_j}(A_1)$$
$$\Gamma_1^k(i_0 \ldots i_{k-1}, r_1) = \{(\emptyset y_0 \ldots y_l \ldots y_{k-1}) | \forall l, 0 \leq l \leq k-1, y_l = h_1(\delta_1(i_0 \ldots i_l, r_1))\}$$
$$\Gamma_2^k(i_0 \ldots i_{k-1}, r_2) = \{(\emptyset y_0 \ldots y_l \ldots y_{k-1}) | \forall l, 0 \leq l \leq k-1, y_l = h_2(\delta_2(i_0 \ldots i_l, r_2))\}$$
$$[\Gamma_1^k(i_0 i_1 \ldots i_{k-1}, r_1) = \Gamma_2^k(i_0 i_1 \ldots i_{k-1}, r_2)],$$

where, again due to determinism, these sequence sets are singletons. What must be proven, then, is the following:

$$\forall j \in \{1, \ldots, |O|\}, \mathcal{L}_{m_j}(A_1) \subseteq \mathcal{L}_{m_j}(A_2) \Leftrightarrow \forall k \geq 1, \forall i_0 i_1 \ldots i_{k-1} \in \bigcup_{j \in \{1, \ldots, |O|\}} \mathcal{L}_{m_j}(A_1)$$

$$\Gamma_1^k(i_0 \dots i_{k-1}, r_1) = \{(\emptyset y_0 \dots y_l \dots y_{k-1}) | \forall l, 0 \le l \le k-1, y_l = h_1(\delta_1(i_0 \dots i_l, r_1))\}$$
$$\Gamma_2^k(i_0 \dots i_{k-1}, r_2) = \{(\emptyset y_0 \dots y_l \dots y_{k-1}) | \forall l, 0 \le l \le k-1, y_l = h_2(\delta_2(i_0 \dots i_l, r_2))\}$$

$$[\Gamma_1^k(i_0 i_1 \dots i_{k-1}, r_1) = \Gamma_2^k(i_0 i_1 \dots i_{k-1}, r_2)].$$

Sufficiency: Choose some input sequence $t = i_0 i_1 \dots i_{m-1} \in \mathcal{I}^m(r_1)$; hence, by Mealy-to-Moore model conversion we also have $t \in \mathcal{L}(A_1)$. Since $F_1 \subseteq_{\mathcal{B}} F_2$ we have:

$$\begin{aligned}(\emptyset y_{\omega_0^1} \dots y_{\omega_l^1} \dots y_{\omega_m^1}) &= \Gamma_1^m(i_0 i_1 \dots i_{m-1}, r_1) \\ &= \Gamma_2^m(i_0 i_1 \dots i_{m-1}, r_2) = (\emptyset y_{\omega_0^2} \dots y_{\omega_l^2} \dots y_{\omega_m^2}).\end{aligned}$$

Since A_1 and A_2 are deterministic, only the last generated output of the sequence needs to be considered. Because the output sequences are equal we know that $y_{\omega_m^1} = y_{\omega_m^2}$. Assume that $y_j = y_{\omega_m^1}$, then $t \in \mathcal{L}_{m_j}(A_1)$. Furthermore, since $y_j = y_{\omega_m^1} = y_{\omega_m^2}$ we also have $t \in \mathcal{L}_{m_j}(A_2)$. From this we conclude $\forall j \in \{1, \dots, |O|\}$, $\mathcal{L}_{m_j}(A_1) \subseteq \mathcal{L}_{m_j}(A_2)$.

Necessity: (By induction on length of sequences). We want to show that if $\mathcal{L}_{m_j}(A_1) \subseteq \mathcal{L}_{m_j}(A_2)$ for all j, then $\forall k \ge 1, \forall i_0 i_1 \dots i_{k-1} \in \bigcup_{j \in \{1,\dots,|O|\}} \mathcal{L}_{m_j}(A_1)$ the following holds:

$$\Gamma_1^k(i_0 i_1 \dots i_{k-1}, r_1) = \Gamma_2^k(i_0 i_1 \dots i_{k-1}, r_2).$$

Every output sequence contains $\emptyset$ as its initial element, so when no input has occurred A_1 and A_2 generate the same $\emptyset$. Now, for $k = 1$, choose $i_0 \in \bigcup_{j \in \{1,\dots,|O|\}} \mathcal{L}_{m_j}(A_1)$. Assume $i_0 \in \mathcal{L}_{m_{j0}}(A_1)$, then the state of A_1 reached following the execution of i_0 is m_{j_0}-marked, and

$$\Gamma_1^1(i_0, r_1) = (\emptyset y_{j_0}).$$

Since $\mathcal{L}_{m_{j0}}(A_1) \subseteq \mathcal{L}_{m_{j0}}(A_2)$ this implies

$$\Gamma_2^1(i_0, r_2) = (\emptyset y_{j_0}) = \Gamma_1^1(i_0, r_1).$$

The induction hypothesis is that for $k = n$, $\forall i_0 i_1 \dots i_{n-1} \in \bigcup_{j \in \{1,\dots,|O|\}} \mathcal{L}_{m_j}(A_1)$ we have

$$\Gamma_1^n(i_0 i_1 \dots i_{n-1}, r_1) = \Gamma_2^n(i_0 i_1 \dots i_{n-1}, r_2).$$

Choose some $t' = i_0 i_1 \dots i_n \in \bigcup_{j \in \{1,\dots,|O|\}} \mathcal{L}_{m_j}(A_1)$, and let r_1' be the state of A_1 reached following the execution of t'. Because A_1 is deterministic we have

$$\begin{aligned}\Gamma_1^{n+1}(i_0 i_1 \dots i_{n-1} i_n, r_1) &= (\emptyset y_{j_0} \dots y_{j_{n-1}} y_{j_n}) \\ &= (\emptyset y_{j_0} \dots y_{j_{n-1}}) h_1(r_1') \\ &= \Gamma_1^n(i_0 i_1 \dots i_{n-1}, r_1) h_1(r_1').\end{aligned} \tag{4}$$

Let r_2' be the state reached in A_2 following the execution of t'. Because $t' \in \mathcal{L}_{m_{jn}}(A_1) \subseteq \mathcal{L}_{m_{jn}}(A_2)$ we know

$$h_2(r_2') = y_{j_n} = h_1(r_1'). \tag{5}$$

Equations 4, 5, and the induction hypothesis yield

$$\Gamma_1^{n+1}(i_0 i_1 \dots i_{n-1} i_n, r_1) = \Gamma_1^n(i_0 i_1 \dots i_{n-1}, r_1) h_1(r_1')$$

$$
\begin{aligned}
&= \Gamma_2^n(i_0 i_1 \ldots i_{n-1}, r_2) h_1(r_1') \\
&= \Gamma_2^n(i_0 i_1 \ldots i_{n-1}, r_2) h_2(r_2') \\
&= \Gamma_2^{n+1}(i_0 i_1 \ldots i_{n-1} i_n, r_2)
\end{aligned}
$$

which completes the induction step and the proof. ■

A family of marked languages is called a *multitrace* set in (Arnold, 1990) where the state parameters of the multitrace set are the marking classes used here. The Language Inclusion Lemma, simply put, says that when the models considered are deterministic the *multitrace* sets with elements of the form

$$i_0 i_1 \ldots i_{k-1} O_{k-1},$$

representing a family of marked languages, contain the same modeling information as *extended trace* sets with elements of the form

$$i_0 O_0 i_1 O_1 \ldots i_{k-1} O_{k-1},$$

representing the behavior of a FSM. See [(Arnold, 1994), pages 148–158] for a thorough discussion and generalization of trace equivalences.

In addition to the Language Inclusion Lemma, which will allow the use of marked languages to determine behavioral inclusion, there is yet another relationship which links behaviors that are matchable in the presence of disturbances to the controllability of their associated m_{os}-marked language. This relationship will be described in detail later in the proofs of Theorem 7.1 in Section 7.2 and Theorem 7.2 in Section 7.3.

7.2. Solving BSCP as FSM Model Matching

This section describes how the model matching formalism proposed by (DiBenedetto *et al*., 1994, 1995, 1996) can be used to solve the Basic Supervisory Control Problem. More specifically, it is shown that the BSCP posed in the framework of supervisory control can be translated to the I/O FSM modeling framework of (DiBenedetto *et al*., 1994, 1995, 1996) and solved as the SMMP–D–MB.

THEOREM 7.1 *Let H and G be as in Definition 3.1, and let M and M_1 be such that $H \overset{IO}{\longmapsto} M$ and $G \overset{IO}{\longmapsto} M_1$; hence $V = U$ and $W = \Sigma_{uc}$. Solve the SMMP–D–MB given M and M_1 for the solution M_2 such that $M_1 \circ M_2 =_{\mathcal{B}} \hat{M}$. Let $H^\dagger$ be the solution to the BSCP, i.e., $\mathcal{L}(H^\dagger) = \mathcal{L}(H \times G)^\uparrow$, and $H^\dagger \overset{IO}{\longmapsto} M^\dagger$. Furthermore, let $\hat{M} \overset{sct}{\longmapsto} \hat{H}$. Then $\hat{M} =_{\mathcal{B}} M^\dagger$, and $\mathcal{L}(\hat{H}) = \mathcal{L}(H^\dagger)$.*

Proof 7.1: Each state of $\hat{H}$ can be associated with a state of $\hat{M}$ by identifying states reached in both by the same input sequence. Because $V = U$ and W is common among the deterministic machines, we can use traces to uniquely identify states. Let t be an input sequence in the behavior represented by $\hat{M}$, and denote the state of $\hat{M}$ reached following the

execution of t by (s_1^t, s_2^t) where s_1^t is the state of the FSM M_1 reached following the execution of t. Let the corresponding state in $\hat{H}$ be denoted by (x_H^t, x_G^t) where x_H^t corresponds to the state of H reached by the input sequence t and x_G^t corresponds to the state of G reached by the input sequence t.

$\hat{M}$ being matchable in the presence of disturbances implies that for all states of $\hat{M}$

$$\mathcal{W}((s_1^t, s_2^t)) = \mathcal{W}(s_1^t). \tag{6}$$

For the associated states in $\hat{H}$ and G we have

$$\Sigma_{uc}(x_{\hat{H}}^t) = \Sigma_{uc}((x_H^t, x_G^t)) = \Sigma_{uc}(x_G^t), \tag{7}$$

where $\Sigma_{uc}(x)$ represents the set of uncontrollable events defined at state x, i.e., $act_*(x) \cap \Sigma_{uc}$.

Clearly, Eqn. 7 defines a bisimulation relation between $\hat{H}$ and G with respect to the set of uncontrollable events. From Theorem 3.1 it follows that $\mathcal{L}(\hat{H})$ is controllable with respect to $\mathcal{L}(G)$ and $\Sigma_{uc} = W$; hence,

$$\mathcal{L}(\hat{H}) \subseteq \mathcal{L}(H^\dagger),$$

and by the Language Inclusion Lemma we have

$$\hat{M} \subseteq_{\mathcal{B}} M^\dagger.$$

Because $\mathcal{L}_m(H^\dagger) = \mathcal{L}(H^\dagger) \subseteq \mathcal{L}(G) = \mathcal{L}_m(G)$ the Language Inclusion Lemma implies $M^\dagger \subseteq_{\mathcal{B}} M_1$, and $M^\dagger$ does not require M_1 to do anything that it is unable to do. This "feasibility" of $M^\dagger$ allows us to claim that a controller, $M_2^\dagger$, exists such that $M_1 \circ M_2^\dagger =_{\mathcal{B}} M^\dagger$ (consider a controller which simply passes inputs to outputs identically iff that input is defined in $M^\dagger$). By Theorem 3.1, the controllability of $\mathcal{L}(H^\dagger)$ defines a bisimulation relation with respect to $\mathcal{L}(G)$ and Σ_{uc}; hence, by a similar argument as in Eqns. 6 and 7, $M^\dagger$ is matchable in the presence of disturbances. By assumption, $\hat{M}$ contains all other behaviors that are matchable in the presence of disturbances; hence,

$$M^\dagger \subseteq_{\mathcal{B}} \hat{M}, \text{ and } \mathcal{L}(H^\dagger) \subseteq \mathcal{L}(\hat{H}).$$

Therefore $\mathcal{L}(H^\dagger) = \mathcal{L}(\hat{H})$ which, for deterministic FSMs, yields $M^\dagger =_{\mathcal{B}} \hat{M}$. ■

The conjecture of (DiBenedetto *et al*., 1994, 1995, 1996) that the BSCP can be solved in the I/O FSM model matching framework has been verified. This is insufficient, however, to conclude that the strong model matching paradigm is more general than the feedback loop of supervisory control. Section 7.3 below investigates this further by discussing a means to map an instance of the SMMP (specifically, SMMP–D–MB) to the Ramadge–Wonham SCT paradigm.

7.2.1. An Example of Solving the BSCP as FSM Model Matching

Consider using the I/O FSM model-matching framework to solve the BSCP on the systems shown in Figures 4 and 5. The converted models are shown in Figures 14 and 15 respectively where the output associated with a marked state is "1" (recall $\Sigma_{uc} = \{\sigma_4\}$).

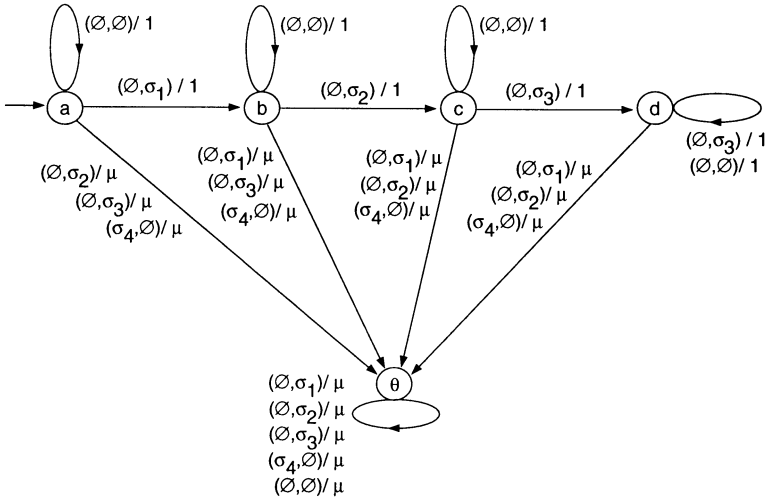

Figure 14. I/O FSM representing desired behavior (cf. Fig. 4).

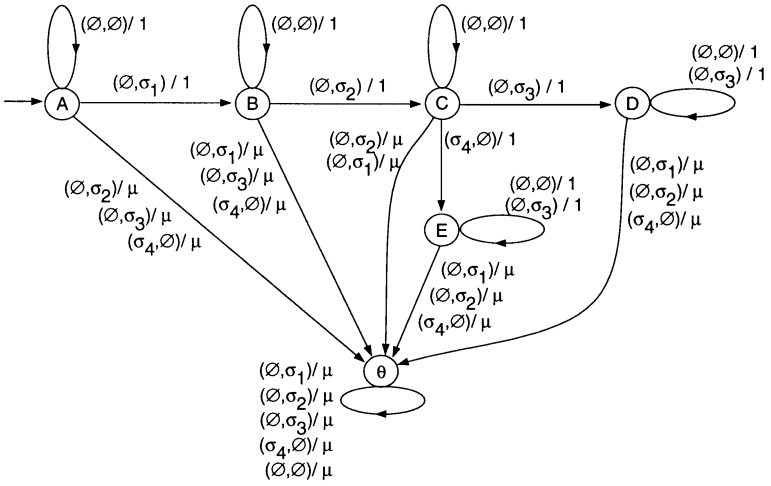

Figure 15. I/O FSM representing plant (cf. Fig. 5).

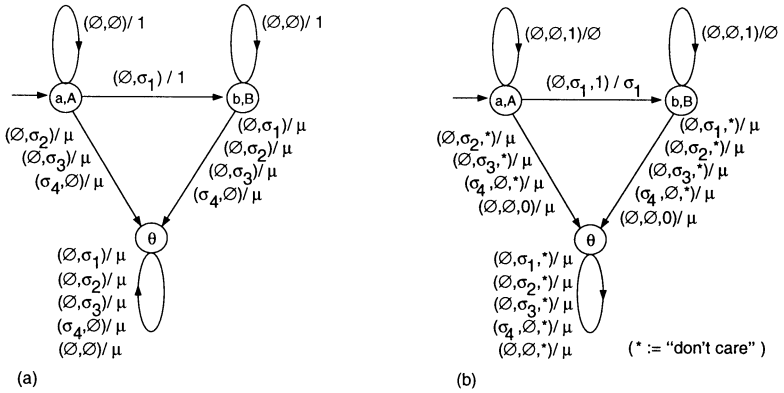

Figure 16. (a) Maximal matchable behavior and (b) associated controller.

Given these two finite-state machines, the maximal matchable behavior in the presence of disturbances and the corresponding controller, M_2, can be found using the method described in (DiBenedetto *et al*., 1996) and are shown in Figure 16. The maximal behavior corresponds to the automaton shown in Figure 6 (up to the completion of the transition function).

7.2.2. Solving BSCP–NB as FSM Model Matching

Although the Basic Supervisory Control Problem for prefix-closed languages can be solved in the realm of finite-state machine model matching, the case of non-prefix-closed languages has not been addressed. Non-prefix-closed languages can lead to blocking solutions in Supervisory Control Theory; hence, a more general problem, the BSCP–NB, is required.

According to the model translation procedures presented in Section 6, an instance of the BSCP–NB translated to the FSM model matching framework (assuming marked states have an output value "1") would require an additional condition such as "all input sequences can always be continued to sequences that produce 1-outputs." As is the case in the SCT framework, an iterative procedure would be required in the FSM model matching framework that removes input sequences that cannot be extended to sequences that end with a 1-output. The algorithm for solving the BSCP–NB in the FSM model matching framework would need to be iterative for the removal of illegal sequences may result in behavior not matchable in the presence of disturbances; thus, the SMMP–D–MB would need to be solved again after each "trimming" operation.

7.3. Solving FSM Model Matching as the BSCP

This section describes how the SMMP–D–MB can be solved as an instance of the BSCP (specifically, the BSCP–NB). Thus, I/O FSM model matching can be solved as a supervisory control problem.

In the previous section where the BSCP is solved in the I/O FSM framework, the set of FSM inputs for M_1 and M were defined over the same alphabets, i.e, $V = U$. When $V \neq U$ (as is often the case when solving the SMMP–D–MB) it is necessary to use the *Sync* automaton to model the I/O FSM composition operator in the SCT paradigm.

Previously, it was shown in the Language Inclusion Lemma that, in deterministic settings, behavioral inclusion for I/O FSMs is equivalent to *marked* language inclusion of corresponding automata; therefore, any useful means of determining "feasible" behavior must maintain deterministic models. Consider, then, the following procedure to generate a DFSA that represents the (not necessarily controllable) set of marked languages corresponding to sub-behavior of a DFSM that can be matched by a DFSM. The procedure is given the acronym DCM meaning "deterministic and consistent Moore automaton."

Procedure: DCM
Step 1 : Given DFSMs M_1 and M where

$$M_1 = ((W, U), Y, S_1, \lambda_1, \gamma_1, s_{1,0})$$
$$M = ((W, V), Y, S, \lambda, \gamma, s_0),$$

generate H and G such that $M \overset{sct}{\longmapsto} H$ (DFSA) and $M_1 \overset{sct}{\longmapsto} G$ (DFSA). Every state in H and G is marked as belonging to the class m_{os}. The class m_{os} holds the meaning "output state". Note that the dead-states of M_1 and M are not present in H and G.

Step 2 : Mark only the initial state of the *Sync* automaton as belonging to the class m_{os}, and form the parallel composition $\tilde{G} = Sync \,||\, G$. Form the parallel composition of H, *Sync* and G: $\tilde{H} = H||\tilde{G}$. In the parallel composition a resulting composite state is m_i-marked if and only if all constituent states are m_i-marked, and a state is m_{os}-marked iff both constituent states are m_{os}-marked. Furthermore, if two states are composed that do not share the same output marking then the resulting state does not have an associated output value (or the output could be considered NULL). Define $\Sigma = W \cup V \cup U$ and $\Sigma_{uc} = W$.

Step 3 : $\forall t \in \mathcal{L}_{m_{os}}(\tilde{H})$, remove all m_{os}-marked states $x_{\tilde{H}} = \delta_{\tilde{H}}(x_{\tilde{H}0}, t)$ from $\tilde{H}$ for which the output is NULL, i.e., $h_{\tilde{H}}(x_{\tilde{H}}) \neq h_{\tilde{G}}(x_{\tilde{G}})$ where $x_{\tilde{G}} = \delta_{\tilde{G}}(x_{\tilde{G}0}, t)$. Perform the "trim" operation (removes states which are not accessible with respect to $x_{\tilde{H}0}$ or coaccessible with respect to the remaining states in $X_{\tilde{H}m_{os}}$), and call the resulting DFSA automaton $\tilde{H}^*$.

Note: $\forall t \in \mathcal{L}_{m_{os}}(\tilde{H}^*)$, there exists a unique output marking class, that is, a state can only produce a single output. This fact results from the following:

1. every transition in M_1 generates an output,

2. $\tilde{G}$ is deterministic, and

3. after H is composed with $\tilde{G}$ the above Step 3 will remove all output states that do not "agree" with the unique output marking following trace t in $\tilde{G}$.

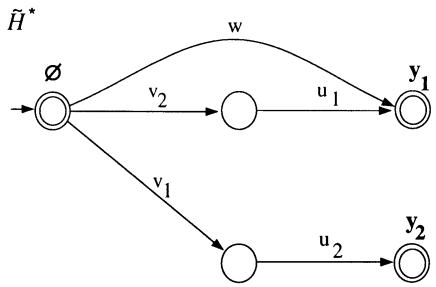

Figure 17. Resulting deterministic Moore automaton with consistent state outputs.

Denote the transformation of DFSM M to the deterministic and consistent Moore automaton $\tilde{H}^*$ (which is dependent on M_1 and $\tilde{G}$) by $(M_1, M) \stackrel{dcm}{\longmapsto} (\tilde{G}, \tilde{H}^*)$. The automaton $\tilde{H}^*$ is called consistent, for every state of $\tilde{H}^*$ reached by some trace of events t has an output class which is consistent with the output class of the state in $\tilde{G}$ reached by t.

As an example of DCM, consider the FSMs M and M_1 of Figure 12. The automata H and G in Figure 12 illustrate Step 1 of the DCM procedure. Composing H and G as described in Step 2 of DCM yields the automaton $\tilde{H}$ shown in Figure 13. The deterministic automaton $\tilde{H}^*$ of Figure 17 illustrates Step 3 of DCM where NULL output m_{os}-marked states are removed and the resulting automaton is trimmed.

Given the ability to translate models from each framework to the other, it is now possible to present the major result of this section.

THEOREM 7.2 *Given DFSMs M_1 and M where*

$$\begin{aligned} M_1 &= ((W, U), Y, S_1, \lambda_1, \gamma_1, r_1) \\ M &= ((W, V), Y, S, \lambda, \gamma, r); \end{aligned}$$

let M_2 be any solution to the SMMP-D-SB, if such a solution exists, i.e.,

$$\begin{aligned} M_1 \circ M_2 &=_{\mathcal{B}} \hat{M}, \text{ and} \\ \hat{M} &\subseteq_{\mathcal{B}} M, \end{aligned}$$

where $\hat{M}$ is a DFSM. Let $H, G, \tilde{G}, \tilde{H}$, and $\tilde{H}^*$ result from $(M_1, M) \stackrel{dcm}{\longmapsto} (\tilde{G}, \tilde{H}^*)$. Define $H^\dagger$ to be the automaton representing the solution to the BSCP–NB given $\tilde{H}^*$, $\tilde{G}$ and $\Sigma_{uc} = W$ with non-blocking behavior determined by m_{os}-marked states, that is, $\mathcal{L}_{m_{os}}(H^\dagger)$ is the supremal controllable sublanguage of $\mathcal{L}_{m_{os}}(\tilde{H}^*)$ with respect to $\mathcal{L}(\tilde{G})$ and $\Sigma_{uc} = W$. Finally, let $P_{W,V}(H^\dagger) \stackrel{IO}{\longmapsto} M^\dagger$. If $M^\dagger$ is nontrivial, then

(i) $M^\dagger \subseteq_{\mathcal{B}} M$,

(ii) A controller, $M_2^\dagger$, can be constructed from $H^\dagger$ such that $M_1 \circ M_2^\dagger =_{\mathcal{B}} M^\dagger$, and $\forall (s_1, s_2^\dagger) \in S^\dagger : \mathcal{W}((s_1, s_2^\dagger)) = \mathcal{W}(s_1)$; hence, $M_2^\dagger$ is a solution to the SMMP-D-SB with respect to M_1 and M.

(iii) $\hat{M} \subseteq_{\mathcal{B}} M^{\dagger}$,

(iv) If $\hat{M}$ *satisfies condition (3) of the SMMP-D-MB problem statement, that is,* $\hat{M}$ *is the largest deterministic behavior that can be matched in the presence of disturbances, then* $\hat{M} =_{\mathcal{B}} M^{\dagger}$.

Proof: Each part of Theorem 7.2 is given particular attention below.
(i) By the construction of $\tilde{H}^*$ and $H^{\dagger}$ we have $\forall j \in \{1, \ldots, |O|\}$:

$$\mathcal{L}_{m_j}(H^{\dagger}) \subseteq \mathcal{L}_{m_j}(\tilde{H}^*) \subseteq \mathcal{L}_{m_j}(\tilde{H}) = \mathcal{L}_{m_j}(H||\tilde{G}) \subseteq \mathcal{L}_{m_j}(P^{-1}_{W,V}(H)).$$

Projecting onto the set $W \cup V$ yields

$$\mathcal{L}_{m_j}(P_{W,V}(H^{\dagger})) = P_{W,V}(\mathcal{L}_{m_j}(H^{\dagger})) \subseteq P_{W,V}(\mathcal{L}_{m_j}(P^{-1}_{W,V}(H))) = \mathcal{L}_{m_j}(H),$$

which, by the Language Inclusion Lemma, implies $M^{\dagger} \subseteq_{\mathcal{B}} M$.

(ii) By the construction of $\tilde{H}^*$ and $H^{\dagger}$ we have $\forall j \in \{1, \ldots, |O|\}$:

$$\mathcal{L}_{m_j}(H^{\dagger}) \subseteq \mathcal{L}_{m_j}(\tilde{H}^*) \subseteq \mathcal{L}_{m_j}(\tilde{G}).$$

Projecting onto the alphabet of M_1, $W \cup U$, yields

$$\forall j,\ P_{W,U}(\mathcal{L}_{m_j}(H^{\dagger})) \subseteq P_{W,U}(\mathcal{L}_{m_j}(\tilde{H}^*)) \subseteq P_{W,U}(\mathcal{L}_{m_j}(\tilde{G})) = \mathcal{L}_{m_j}(G);$$

hence, $\mathcal{L}_{m_{os}}(H^{\dagger})$ contains only traces which correspond to input/output sequences that are possible in the plant, M_1. To show that there exists $M_2^{\dagger}$ such that $M_1 \circ M_2^{\dagger} =_{\mathcal{B}} M^{\dagger}$ consider the following construction using $H^{\dagger} = (X_{H^{\dagger}}, \Sigma_{H^{\dagger}}, \delta_{H^{\dagger}}, act_{H^{\dagger}}, x_{H0^{\dagger}}, \mathcal{X}_{H_m^{\dagger}})$:

Define $F = ((W, V, Y), (W, U), S_F, \lambda_F, \gamma_F, r_F)$, where

$$S_F \subseteq X_{H^{\dagger}}$$

$$r_F = x_{H0^{\dagger}}$$

$$\lambda_F((w, v, y), s) = \begin{cases} \delta_{H^{\dagger}}(t, s) & \text{if } [(t = w) \vee (\exists u \in U \text{ such that } t = vu)] \wedge \\ & [y = h_{H^{\dagger}}(\delta_{H^{\dagger}}(t, s))] \\ \theta_F & \text{otherwise} \end{cases}$$

$$\gamma_F((w, v, y), s, s') = \begin{cases} (\emptyset, u) & \text{if } [w = \emptyset] \wedge [\delta_{H^{\dagger}}(vu, s) = s'] \wedge \\ & [y = h_{H^{\dagger}}(s')] \\ (w, \emptyset) & \text{if } [v = \emptyset] \wedge [\delta_{H^{\dagger}}(w, s) = s'] \wedge \\ & [y = h_{H^{\dagger}}(s')] \\ (\mu, \mu) & \text{if } s' = \theta_F \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Examining the definition of FSM composition, it is evident that $M_1 \circ F =_{\mathcal{B}} M^{\dagger}$, and $M_2^{\dagger} = F$ is a compensator which produces the desired result.

Consider now, the implications of the controllability of $\mathcal{L}_{m_{os}}(H^\dagger)$ with respect to $\mathcal{L}(\tilde{G})$ and $\Sigma_{uc} = W$. By Theorem 3.1 and Definition 2.8 the following equalities are true $\forall x_{H^\dagger} = (x_H, x_{Sync}, x_G) \in X_{H^\dagger}$:

$$\begin{aligned}\Sigma_{uc}(x_{H^\dagger}) &= \Sigma_{uc}((x_H, x_{Sync}, x_G)) \\ &= \Sigma_{uc}((x_{Sync}, x_G)) \\ &= \Sigma_{uc}(x_G).\end{aligned}$$

Because $P_{W,V}(H^\dagger) \overset{IO}{\longmapsto} M^\dagger$ and $M^\dagger =_{\mathcal{B}} M_1 \circ M_2^\dagger$, the above equalities imply that $\forall (s_1, s_2^\dagger) \in S^\dagger : \mathcal{W}((s_1, s_2^\dagger)) = \mathcal{W}(s_1)$; hence, $M_2^\dagger$ is a solution to the SMMP-D-SB with respect to M_1 and M.

(iii) Define $\tilde{H}^*_{W,V} = P_{W,V}(\tilde{H}^*)$. Let $\hat{M} \overset{sct}{\longmapsto} \hat{H}$ and $\tilde{H}^*_{W,V} \overset{IO}{\longmapsto} \tilde{M}^*_{W,V}$. By the Language Inclusion Lemma, $\hat{M} \subseteq_{\mathcal{B}} \tilde{M}^*_{W,V}$ if and only if $\mathcal{L}_{m_j}(\hat{H}) \subseteq \mathcal{L}_{m_j}(\tilde{H}^*_{W,V}), \forall j$. We will show containment for each marked language by induction on the length of traces in $\mathcal{L}_{m_{os}}(\hat{H})$.

(Basis of induction): For trace t such that $|t| = 0$, $t = \epsilon$ and $x_{\hat{H}_0} = \delta_{\hat{H}}(x_{\hat{H}_0}, t)$. Furthermore, $x_{\tilde{H}^*_{WV0}} = \delta_{\tilde{H}^*_{WV}}(x_{\tilde{H}^*_{WV0}}, t)$, and by construction $h_{\hat{H}}(x_{\hat{H}_0}) = h_{\tilde{H}^*_{WV}}(x_{\tilde{H}^*_{WV0}}) = \emptyset$ which we can associate with its own marking class $m_\emptyset$. Thus, $t \in \mathcal{L}_{m_\emptyset}(\hat{H})$ and $t \in \mathcal{L}_{m_\emptyset}(\tilde{H}^*_{W,V})$.

(Inductive hypothesis): Assume for all traces $t \in \mathcal{L}(\hat{H})$ of length $|t| = n$, $[t \in \mathcal{L}_{m_i}(\hat{H})] \Rightarrow [t \in \mathcal{L}_{m_i}(\tilde{H}^*_{W,V})]$.

(Inductive step): Now, we show if $t\sigma \in \mathcal{L}_{m_j}(\hat{H})$ ($|t\sigma| = n + 1$), then $t\sigma \in \mathcal{L}_{m_j}(\tilde{H}^*_{W,V})$:
Let $t \in \mathcal{L}_{m_i}(\hat{H})$ be of length n; hence, $t \in \mathcal{L}_{m_i}(\tilde{H}^*_{W,V})$. Choose σ such that $t\sigma \in \mathcal{L}_{m_j}(\hat{H})$, and let $x_{\hat{H}} = \delta_{\hat{H}}(x_{\hat{H}0}, t)$. $t \in \mathcal{L}_{m_i}(\tilde{H}^*_{W,V})$ implies there exists $t' \in P^{-1}_{W,V}(\mathcal{L}_{m_i}(\tilde{H}^*_{W,V}))$ such that $t' \in \mathcal{L}_{m_i}(\tilde{H}^*)$. Let $(x_H, x_{\text{Sync}}, x_G) = \delta_{\tilde{H}^*}((x_{H0}, x_{\text{Sync},0}, x_{G0}), t')$, and identify:

1. x_H with its corresponding state s in M, and
2. x_G with its corresponding state s_1 in M_1.

Since $t \in \mathcal{L}_{m_i}(\tilde{H}^*_{W,V})$ we also know that $h_H(x_H) = h_G(x_G) = y_i$. Now,

$$\begin{aligned}t\sigma \in \mathcal{L}_{m_j}(\hat{H}) &\Rightarrow x'_{\hat{H}} = \delta_{\hat{H}}(x_{\hat{H}}, \sigma) \text{ and } h_{\hat{H}}(x'_{\hat{H}}) = h_H(x'_H) = y_j \\ &\Rightarrow \hat{s}' = \hat{\lambda}(\sigma, \hat{\lambda}(t, \hat{r})) \text{ and } \hat{\gamma}(\sigma, \hat{\lambda}(t, \hat{r}), \hat{s}') = y_j.\end{aligned}$$

There are two possible cases concerning the identity of σ:

$$\begin{aligned}
\text{Case 1: } \sigma \in W \;&\Rightarrow\; \exists s_1' = \lambda_1(\sigma, s_1) \text{ and } \gamma_1(\sigma, s_1, s_1') = y_j \\
&\Rightarrow\; \exists x_G' = \delta(x_G, \sigma) \text{ and } h_G(x_G') = h_H(x_H') = y_j \\
&\Rightarrow\; (x_H', x_{\text{Sync},0}, x_G') = \delta_{\tilde{H}^*}((x_H, x_{\text{Sync},0}, x_G), \sigma) \text{ and} \\
&\quad\; h_{\tilde{H}^*}((x_H', x_{\text{Sync},0}, x_G')) = y_j \\
&\Rightarrow\; t'\sigma \in \mathcal{L}_{m_j}(\tilde{H}^*) \\
&\Rightarrow\; t\sigma \in \mathcal{L}_{m_j}(\tilde{H}^*_{W,V}),
\end{aligned}$$

$$\begin{aligned}
\text{Case 2: } \sigma \in V \;&\Rightarrow\; \text{(by FSM composition) } \exists u \in U \text{ such that} \\
&\quad\; \exists s_1' = \lambda_1(u, s_1) \text{ and } \gamma_1(u, s_1, s_1') = y_j \\
&\Rightarrow\; \exists x_G' = \delta(x_G, u) \text{ and } h_G(x_G') = h_H(x_H') = y_j \\
&\Rightarrow\; (x_H', x_{\text{Sync},0}, x_G') = \delta_{\tilde{H}^*}((x_H, x_{\text{Sync},0}, x_G), \sigma u) \text{ and} \\
&\quad\; h_{\tilde{H}^*}((x_H', x_{\text{Sync},0}, x_G')) = y_j \\
&\Rightarrow\; t'\sigma u \in \mathcal{L}_{m_j}(\tilde{H}^*) \\
&\Rightarrow\; t\sigma \in \mathcal{L}_{m_j}(\tilde{H}^*_{W,V}),
\end{aligned}$$

which completes the proof by induction that $\mathcal{L}_{m_j}(\hat{H}) \subseteq \mathcal{L}_{m_j}(\tilde{H}^*_{W,V}), \forall j$.

Define a submachine of $\tilde{H}^*$, H_{Req}, such that

$$\mathcal{L}_{m_{os}}(H_{\text{Req}}) = P^{-1}_{W,V}(\mathcal{L}_{m_{os}}(\hat{H})) \cap \mathcal{L}_{m_{os}}(\tilde{H}^*).$$

Since $\mathcal{L}_{m_{os}}(\hat{H}) \subseteq P_{W,V}(\mathcal{L}_{m_{os}}(\tilde{H}^*))$ it is evident[16] that $P_{W,V}(\mathcal{L}_{m_{os}}(H_{\text{Req}})) = \mathcal{L}_{m_{os}}(\hat{H})$, and

$$\mathcal{L}_{m_{os}}(H_{\text{Req}}) \subseteq \mathcal{L}_{m_{os}}(\tilde{H}^*).$$

For brevity, make the following associations for all commonly defined traces $t \in \mathcal{L}(\tilde{H}^*)$:

1. Associate $x_H^t = \delta_H(x_{H0}, P_{W,V}(t))$ in H with $s^t = \lambda(P_{W,V}(t), r)$ in M.
2. Associate $x_G^t = \delta_G(x_{G0}, P_{W,U}(t))$ in G with $s_1^t = \lambda_1(P_{W,U}(t), r_1)$ in M_1.
3. Associate $x_{\hat{H}}^t = \delta_{\hat{H}}(x_{\hat{H}0}, P_{W,V}(t))$ in $\hat{H}$ with $\hat{s}^t = \hat{\lambda}(P_{W,V}(t), \hat{r}) = (s_1^t, s_2^t)$ in $\hat{M}$.

Note that H_{Req}, $H^\dagger$ and $\tilde{H}^*$ are all defined on the common automata H, G and $Sync$, so for commonly defined traces we have:

$$x^t_{H_{Req}} = x^t_{H^\dagger} = x^t_{\tilde{H}^*} = (x_H^t, x_{\text{Sync}}^t, x_G^t) = \delta_{\tilde{H}^*}((x_{H0}, x_{\text{Sync},0}, x_{G0}), t).$$

Furthermore, since $\hat{M} \subseteq_{\mathcal{B}} M$ we can associate $\hat{s}^t$ with s^t (which is associated with x_H^t) for all commonly defined traces t.

Now, consider the implications of $\hat{M}$ being matchable in the presence of disturbances:

$$\begin{aligned}\forall(s_1^t, s_2^t) \in \hat{S}, \mathcal{W}((s_1^t, s_2^t)) = \mathcal{W}(s_1^t) \;&\Rightarrow\; \forall x_{\hat{H}}^t = (x_H^t, x_G^t) \in X_{\hat{H}},\\ &\quad \Sigma_{uc}(x_{\hat{H}}^t) = \Sigma_{uc}(x_G^t)\\ &\Rightarrow\; \forall(x_H^t, x_G^t) \in X_{\hat{H}},\\ &\quad \Sigma_{uc}((x_H^t, x_G^t)) = \Sigma_{uc}(x_G^t)\\ &\Rightarrow\; \forall(x_H^t, x_{\text{Sync}}^t x_G^t) \in X_{H_{Req}},\\ &\quad \Sigma_{uc}((x_H^t, x_{\text{Sync}}^t, x_G^t) = \Sigma_{uc}(x_{\text{Sync}}^t, x_G^t),\end{aligned}$$

which defines a bisimulation relation between H_{Req} and $\tilde{G}$ with respect to Σ_{uc}. Since $\mathcal{L}_{m_{os}}(H_{\text{Req}}) \subseteq \mathcal{L}(\tilde{G})$, Theorem 3.1 yields that $\mathcal{L}_{m_{os}}(H_{\text{Req}})$ is a controllable sublanguage of $\mathcal{L}_{m_{os}}(\tilde{H}^*)$ with respect to $\mathcal{L}(\tilde{G})$ and Σ_{uc}. By supremality, then, we obtain:

$$\mathcal{L}_{m_{os}}(H_{\text{Req}}) \subseteq \mathcal{L}_{m_{os}}(H^\dagger).$$

We also have containment for each marked language because H_{Req} and $H^\dagger$ are both defined on the same state-space of $\tilde{H}^*$, and each state of $\tilde{H}^*$ has a unique output marking. The Language Inclusion Lemma yields:

$$\hat{M} \subseteq_{\mathcal{B}} M^\dagger. \tag{8}$$

(iv) From Eqn. 8 and the result of part (ii) above, we conclude that $M^\dagger$ is matchable (with respect to M and M_1) in the presence of disturbances and contains any other deterministic sub-behavior, $\hat{M}$, that is matchable (with respect to M and M_1) in the presence of disturbances; hence $\hat{M} \subseteq_{\mathcal{B}} M^\dagger$.

If $\hat{M}$ satisfies condition (3) of the SMMP-D-MB problem statement, that is, $\hat{M}$ is the largest deterministic behavior that can be matched in the presence of disturbances, then because M_1 and M are deterministic, $\hat{M}$ contains any other deterministic behavior that can be matched in the presence of disturbances. As shown in part (ii) above, $M^\dagger$ is a deterministic behavior that can be matched in the presence of disturbances, thus $M^\dagger \subseteq_{\mathcal{B}} \hat{M}$ and

$$\hat{M} =_{\mathcal{B}} M^\dagger. \qquad \blacksquare$$

Given DFSMs M_1 and M, the solution of the SMMP-D-MB as an instance of the BSCP-NB is summarized as follows:

1. Transform the DFSMs to Moore-type automata, i.e., $M_1 \overset{sct}{\longmapsto} G$ and $M \overset{sct}{\longmapsto} H$.
2. Determine every possible closed-loop behavior by composing the plant, G, with *Sync* to form $\tilde{G}$.
3. Determine the largest legal subset of the desired behavior by using DCM to form $\tilde{H}^*$.

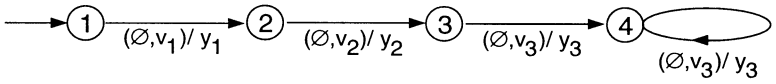

Figure 18. I/O FSM to be matched.

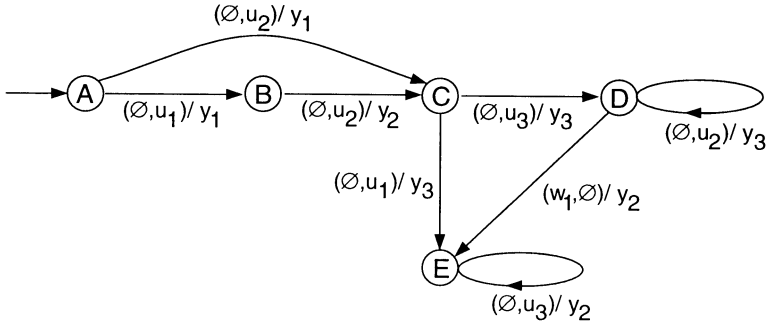

Figure 19. I/O FSM representing plant.

4. Find $H^\dagger$ representing the solution to the BSCP-NB (with respect to $\tilde{H}^*$, $\tilde{G}$, and Σ_{uc}) where non-blocking behavior is determined by m_{os}-marked states.

5. The largest deterministic behavior that can be matched in the presence of disturbances is $P_{w,v}(H^\dagger) \overset{IO}{\longmapsto} M^\dagger$.

6. A controller, $M_2^\dagger$, that realizes $M^\dagger$ can be constructed from $H^\dagger$ (as described in the proof of Theorem 7.2).

It is important to emphasize that the construction of the controller $M_2^\dagger$ is of polynomial complexity as $M_2^\dagger$ is built from $H^\dagger$, and thus it does *not* involve the deterministic projected automaton $P_{w,v}(H^\dagger)$ for which the construction is exponential in the worst case.

The above item 4 uses the relationship between controllability of a language and bisimulation with respect to uncontrollable events discussed in Theorem 3.1.

7.3.1. An Example of Solving FSM Model Matching as the BSCP–NB

Using the I/O FSMs shown in Figures 18 and 19, the previous results are applied to find the controller that generates the maximal matchable behavior in the presence of disturbances (the *dead state* is not shown).

Given the two state diagrams of Figures 18 and 19, the first step of the procedure is to generate the corresponding Moore state diagrams with the appropriate markings. The resulting two Moore machines are shown in Figures 20 and 21.

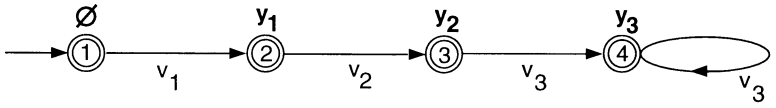

Figure 20. Moore machine, H, to be matched (cf. Fig. 18).

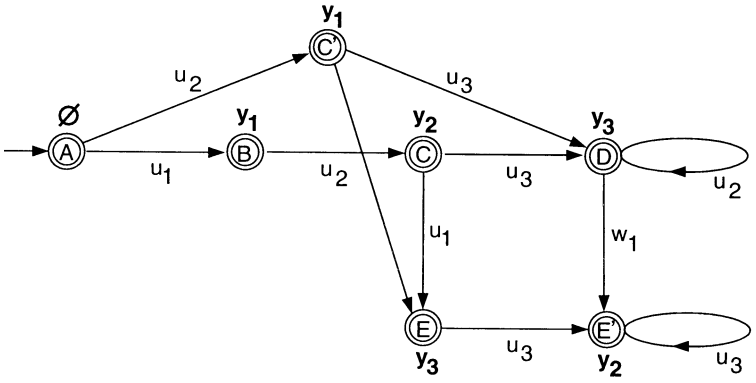

Figure 21. Moore machine, G, representing plant (cf. Fig. 19).

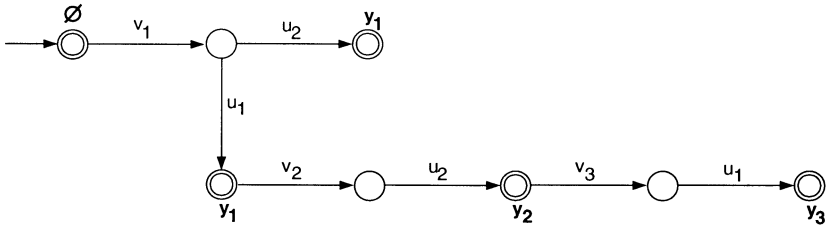

Figure 22. Maximal matchable behavior with disturbances (Solution to BSCP–NB).

The two Moore type machines are modified as described in DCM to form $\tilde{H}^*$ and $\tilde{G}$ which are then sent to BISIM–BSCP–NB to solve the BSCP–NB with the markings representing output states. The solution to the BSCP–NB is shown in Figure 22. As can be seen in Figure 22, the m_{os}-marked states of the controller alternate revealing that system outputs occur only after a command input in V is translated to a control input U. The controller corresponding to Figure 22 which solves the SMMP–D–MB is shown in Figure 23. Had the SMMP–D–MB been solved in the I/O FSM framework, the resulting controller would

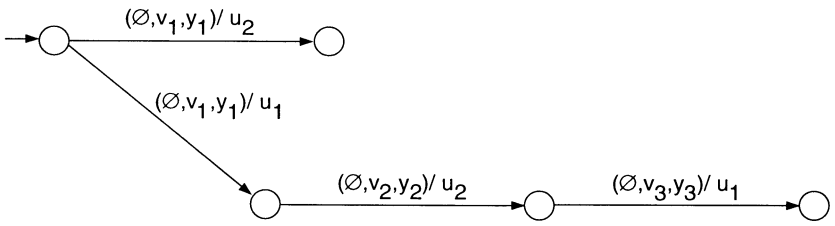

Figure 23. Solution to SMMP–D–MB (cf. Fig. 22).

have been precisely that shown in Figure 23 with, of course, a completed transition function to a "dead" state.

7.4. Discussion of FSM Model Matching and SCT

As shown in Subsections 7.2 and 7.3, when the finite-state machines or automata representing the "plant" and reference model are deterministic the BSCP can be solved as an instance of SMMP–D–MB in the I/O FSM framework, and the SMMP–D–MB can be solved as an instance of the BSCP–NB for a family of marked languages. Thus, for the problems addressed, no framework is more general than the other.[17]

The ability to compare model matching and supervisory control results from being able to associate language controllability to bisimulation and bisimulation to "behavioral controllability" (the matchability of a behavior in the presence of disturbances). For each problem, a bisimulation relation with respect to the uncontrollable occurrences can be found which constrains the set of states that can be paired. The particulars of each framework, such as multiple transition labels, are manipulated so that each type of problem can be solved using a bisimulation relation.

An important feature of the Strong I/O FSM Model Matching Problem is that it is posed in a semantics for which the notion of equivalence is finer than language equivalence. FSM equivalence requires that not only do traces generate the same final output, but every output along the trace must match also. This notion of equivalence is somewhat analogous to Ready Trace equivalence (see Appendix A) which requires the active event sets to match along traces. Supervisory Control Theory is posed in the language semantics; thus, the BSCP has little meaning in nondeterministic settings requiring more detailed semantics. In this way, I/O FSM Model Matching can be compared to SCT: I/O FSM Model Matching has the *potential* to deal with processes described in both deterministic and nondeterministic settings without altering the meaning of equivalence, while SCT (in its original setting) deals with equivalence at the language level which does not extend to nondeterministic settings where automata "branching structure" is important.

Strong I/O FSM Model Matching does not presently deal with issues of blocking; furthermore, it is questionable as to whether the FSM composition operator can deal with

partial observations. The ability to handle issues of blocking and partial observations are significant contributors to the elegance and utility of Supervisory Control Theory, and it has been shown here that using multiple marked languages allows I/O modeling issues to be addressed by SCT.

It is important to note, however, that the generality of the two problems has little to do with the fact that I/O FSMs have multiple labels on each transition while the automata used to represent marked languages in supervisory control have one, i.e., Mealy machines and Moore machines are equivalent in their ability to express behavior. The generality, in the sense described, is derived solely from the semantics in which the problem was posed and solved.

8. Conclusions

In this paper concepts from a different semantics, bisimulation, were used to solve the Basic Supervisory Control Problem in deterministic settings. It was shown that, given language inclusion of deterministic automata, controllability of languages is the same as bisimulation of automata with respect to uncontrollable events. Further, these bisimulation relations can be found by partition refinement for which there exist very efficient algorithms. One such algorithm by (Fernandez, 1990), based on Paige and Tarjan's famous algorithm (Paige and Tarjan, 1987), finds the coarsest partition of a family of binary relations on a set, S, with complexity $O(c\ m \log(n))$ where m is the size of the relation, n is the size of S, and c is a bound on the largest image set of any element of the relation. This algorithm is used in a very efficient "partially" on-line solution of the BSCP: given two automata H and G, a partition that induces the greatest bisimulation relation, Φ_{uc}, with respect to Σ_{uc} is found in $O(|\Sigma_{uc}|(n_H + n_G)\log(n_H + n_G))$ time and $O(n_H + n_G)$ space. The on-line supervisor maintains only a copy of G, H, and the partition representing Φ_{uc} and performs $O(|\Sigma_c|)$ "1-step-lookahead" operations at each current state. Thus, the size of the on-line supervisor is $O(|\Sigma|(n_H + n_G))$ (compare to $O(|\Sigma|(n_H n_G))$).

The Supervisory Control Theory initiated by (Ramadge and Wonham, 1989) was carefully compared to the Strong I/O FSM Model Matching Theory of (DiBenedetto *et al.*, 1994, 1995, 1996). It was formally established that, in deterministic settings, the "Strong I/O FSM Model Matching with Measurable Disturbances to A Maximal Set of Reference Behaviors Problem" can be solved by mapping it to an instance of the "Basic Supervisory Control Problem" and vice-versa. Strong I/O FSM Model Matching does not presently deal with blocking behavior or partial observations; however, several important relationships between the two paradigms were exposed.

The generality of I/O FSM Model Matching was compared to that of Supervisory Control. Being posed in a "stronger" semantics will allow nondeterministic problems to be solved in the I/O FSM Model Matching framework without altering the meaning of machine equivalence; however, all of the problems presented in (DiBenedetto *et al.*, 1994, 1995, 1996) are based on deterministic "plants". Supervisory Control Theory bases equivalence on languages, and therefore, must be altered to handle nondeterministic settings; recent work in that regard can be found in (Fabian, 1995, Heymann and Lin, 1996, Inan, 1994, Overkamp, 1997, Shayman and Kumar, 1995).

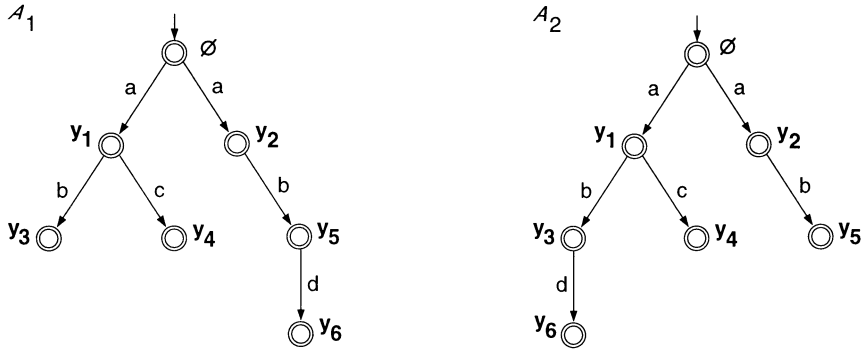

Figure 24. Example nondeterministic automata.

A Investigation of The Language Inclusion Lemma in Nondeterministic Settings

Throughout this work it has been stressed that deterministic models be used for comparisons with Supervisory Control Theory. The fundamental reason for requiring deterministic automata is that SCT was originally proposed in the language semantics, and language equivalence is the weakest requirement that is necessary and sufficient for deterministic automata equivalence.

DiBenedetto *et al*. introduce a notion of equivalence that (with respect to the converted marked automata) in addition to requiring the languages of two automata to be the same, also requires that the sequences of visited marked states be the same. Consider the two automata A_1 and A_2 shown in Figure 24. Every marked language of A_1 and A_2 is equivalent; however, the two automata are not behaviorally equivalent because

$$\Gamma_1^3(abd, x_{A_{1,0}}) = \{(\emptyset y_2 y_5 y_6)\} \neq \{(\emptyset y_1 y_3 y_6)\} = \Gamma_2^3(abd, x_{A_{2,0}}),$$

that is, although the final output marking of the state reached by the trace abd matches in both automata, the sequence of markings visited along the trace do not match. Hence, the Language Inclusion Lemma (LIL) presented in Section 7.1 does not hold.

The above example demonstrates that the LIL is not true when both automata are nondeterministic. The following question can be asked: "can the LIL be used when only one of the automata is nondeterministic?" For insight into the answer to this question consider the automaton A_1' shown in Figure 25. Although every marked language of A_1' is contained by those of A_2, we still do not have $A_1' \subseteq_{\mathcal{B}} A_2$ for the same reason as above. Hence, the Language Inclusion Lemma does not hold when any of the automata are nondeterministic.

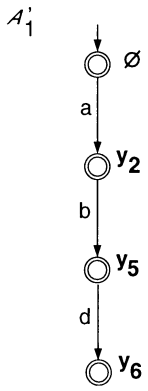

Figure 25. Example deterministic automaton.

Acknowledgements

We gratefully acknowledge Maria DiBenedetto and Alberto Sangiovanni–Vincentelli for the many enlightening discussions on the topics of bisimulation relations and I/O FSM model matching, and for sharing with us a copy of their report (DiBenedetto *et al*., 1996). We also acknowledge relevant comments from the reviewers of this paper, as well as the anonymous reviewers of (Barrett and Lafortune, 1997).

This research was supported in part by NSF grant ECS-9057967 and ARO grant DAAH04-96-1-0377.

Notes

1. In order to make our presentation as self-contained as possible, we are including in Sections 2 and 5 definitions and problem formulations that have appeared in prior works. In all of these cases, references to the literature are given.
2. Reference (Cassandras *et al*., 1995) is not the original reference for the concepts reviewed in this section, but rather a tutorial introduction to these concepts.
3. The deterministic projected automaton is only of theoretical use here in proofs, so we are not concerned with its exponential complexity construction.
4. $\overline{L}$ denotes the prefix-closure of the language L.
5. Reference (Kumar *et al*., 1991) also shows that the bound $O(|\Sigma|(n_H n_G))$ is tight for constructing an automaton that generates $K^{\uparrow}$.
6. In a sense to be made more precise in Section 4; refer to the discussion in Section 4.4.
7. Greatest bisimulation relations will be denoted by Φ.
8. The term "menu" refers to the active event set at a particular state. Menus are not considered "available" as a resource to supervisors in SCT; supervisors only observe traces of events.
9. More specifically, the worst-case running time of the Fernandez algorithm is $O(c\ m \log(n))$ where c is a homogeneous upper bound on the number of times any label appears at any state, i.e., c is the largest image set size due to any event (Fernandez, 1996). Here, the algorithm is only used on *deterministic* automata so $c = 1$.

10. The definition used here is a modified version of that found in (DiBenedetto *et al*., 1995); the notation associated with the output sequences of the FSMs was condensed.
11. This definition has been modified from that presented in (DiBenedetto *et al*., 1995) in terms of notation.
12. A concept defined in (DiBenedetto *et al*., 1995); rougly speaking, the inverse automaton is simply the original state machine using only the output labels on each transition.
13. In (DiBenedetto *et al*., 1995) $\overline{M}$ is referred to as $\tilde{M}$ which is not the same machine as defined earlier in this paper; hence, we use the over-bar to distinguish these FSM here.
14. This is a modified version of a conversion method suggested in (Sangiovanni-Vincentelli, 1995). While the method suggested in (Sangiovanni-Vincentelli, 1995) may work, the method presented here maintains the intuitive notion of assigning the set of uncontrollable events Σ_{uc} to be the set of disturbances W.
15. With some abuse of notation, the curly braces are not shown for singleton outputs, and we will not distinguish an output class from its associated output value.
16. It is straightforward to derive the following **Fact:** Let K, M and L be languages. If $K = P^{-1}(M) \cap L$ where $M \subseteq P(L)$, then $P(K) = M$. Furthermore, $K = P^{-1}[P(K)] \cap L$, i.e., K is normal with respect to P and L.
17. We maintain that the use of multiple marked languages in this paper does not alter Supervisory Control Theory, for existing SCT and language union concepts are used in dealing with the multiple marked language models described herein. That is, our use of multiple sets of marked states is merely a modeling issue within the existing body of Supervisory Control Theory.

References

Arnold, A. 1994. *Finite Transition Systems*. NJ: Prentice Hall.

Baeten, J. C. M., and Weijland, W. P. 1990. Process algebra. *Cambridge Tracts in Theoretical Computer Science* 18.

Barrett, G., and Lafortune, S. 1996. A bisimulation approach to the supervisory control of discrete event systems. *Proc. of 34th Annual Allerton Conference on Communication, Control and Computing*. Allerton Park, IL.

Barrett, G., and Lafortune, S. 1997. Using bisimulation to solve discrete event control problems. *Proc. 1997 American Control Conf.* Albuquerque, NM, pp. 2337–2341.

Bloom, B., Istrail, S., and Meyer, A. 1988. Bisimulation can't be traced: Preliminary report. *Proc. of 15th Annual SIGACT-SIGPLAN Symposium on Principles of Programming Languages*.

Cassandras, C., Lafortune, S., and Olsder, G. 1995. Introduction to the modelling, control and optimization of discrete event systems. *Trends in Control. A European Perspective*. A. Isidori, ed., Springer-Verlag, pp. 217–291.

DiBenedetto, M. D., Saldanha, A., and Sangiovanni-Vincentelli, A. 1994. Model matching for finite state machines. *Proc. of 33rd Conf. Decision and Control*. Lake Buena Vista, FL. pp. 3117–3124.

DiBenedetto, M. D., Saldanha, A., and Sangiovanni-Vincentelli, A. 1995. Strong model matching for finite state machines. *Proc. of 3rd European Control Conference*. Rome, Italy, pp. 2027–2034.

DiBenedetto, M. D., Saldanha, A., and Sangiovanni-Vincentelli, A. 1995. Strong model matching for finite state machines with non-deterministic reference model. *Proc. of 34rd Conf. Decision and Control*. New Orleans, LA. pp. 422–426.

DiBenedetto, M. D., Saldanha, A., and Sangiovanni-Vincentelli, A. 1996. Model matching for finite state machines. Cadence Berkeley Laboratories Technical Report.

Fabian, M. 1995. On object oriented nondeterministic supervisory control. Ph.D. thesis, Chalmers University of Technology.

Fernandez, J. 1990. An implementation of an efficient algorithm for bisimulation equivalence. *Sci. Comput. Programming* 13: 219–236.

Fernandez, J. 1996. Personal communications.

Hadj-Alouane, N. B., Lafortune, S., and Lin, F. 1994. Variable lookahead supervisory control with state information. *IEEE Trans. Automat. Contr*. 39-12: 2398–2410.

Hayes, J. P. 1993. *Introduction to Digital Logic Design*. Reading, MA: Addison-Wesley.

Heymann, M., and Lin, F. 1996. Discrete event control of nondeterministic systems. *Tech. Report # CIS 9601*, Department of Computer Science Technion, Israel Institute of Technology.

Heymann, M., and Meyer, G. 1991. An algebra of discrete event processes. *Tech. Report NASA Memorandum 102848*. NASA, Ames Research Center, Moffett Field, CA.

Inan, K. 1994. Nondeterministic supervision under partial observation. *11th International Conference on Analysis and Optimization of Systems: Discrete Event Systems*. G. Cohen and J. Quadrat, eds., Springer-Verlag, pp. 39–48.

Kohavi, Z. 1978. *Switching and Finite Automata Theory*, 2nd ed. New York: McGraw-Hill.

Kumar, R., Garg, V., and Marcus, S. I. On controllability and normality of discrete-event dynamical systems. *Syst. Contr. Lett*. 17: 157–168.

Overkamp, A. 1997. Supervisory control using failure semantics and partial specifications. *IEEE Trans. Automat. Contr*. 42-4: 498–510.

Paige, R., and Tarjan, R. 1987. Three partition refinement algorithms. *SIAM J. Comput*. 16-6: 973–989.

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

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

Sangiovanni-Vincentelli, A. 1995. Personal communications.

Shayman, M., and Kumar, R. 1995. Supervisory control of nondeterministic systems with driven events via prioritized synchronization and trajectory models. *SIAM J. Control Optim*. 33-2: 469–497.

Thistle, J. G., Malhamé, R. P., Hoang, H. H., and Lafortune, S.1995. Blocking, modularity, and feature interactions in distributed systems. Preprint.

Wonham, W. M., and Ramadge, P. J. 1987. On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim*. 25-3: 637–659.