Model Uncertainty in Discrete Event Systems *

Stanley Young
Vijay K. Garg
Department of Electrical and Computer Engineering,
University of Texas, Austin, TX 78712
stanley@pine.ece.utexas.edu

January 6, 1994

Abstract

Earlier work concerning control of discrete event systems usually assumed that a correct model of the system to be controlled was available. A goal of this wok is to provide an algorithm for determining the correct model from a set of models. The result of the algorithm is a finite language which can be used to test for the correct model or notification that the remaining models cannot be controllably distinguished. We use the finite state machine model with controllable and uncontrollable events presented by Ramadge and Wonham [1].

Keywords: Discrete Event Systems, System Identification
AMS(MOS) subject classification: 93A,93B

*Supported in part by National Science Foundation CCR-911065, and in part by the University Research Institute, University of Texas at Austin.

[1] A preliminary version of this paper appeared as [18]. Another version will appear in SIAM Journal of Control and Optimization.

1 Introduction

A discrete event system (DES) is one which responds to distinct events occurring at asynchronous times [7]. Examples of such systems include computer networks, manufacturing systems, and other dynamic systems which require high level coordinated control. There has been some success recently in developing a theory for the control of such systems (see [13] and the references therein). Most of this work has assumed that an accurate model for the system of interest is available.

The motivation for this work is the desire to control systems in the presence of uncertainty in the model of the system and environment in which the system operates. Part of this work is an extension of learning and inference theory [6, 2, 16] to the domain of discrete event systems. This work is also related to recent results concerning the determination of a system model when certain assumptions are made about the model and type of experiments [14, 15]. In both the learning theory and system determination work, an assumption is that all events are controllable. The uncontrollability of certain events figures prominently in this work. The approach taken in this paper is similar to the approach used for system identification in [17] in that any model which is falsified is dropped from consideration as a correct model.

There are many different types of uncertainty which might occur in a system model. To discuss such uncertainties, a model representation must be chosen. In this work, we investigate uncertainty in a deterministic finite state machine. An example of such uncertainty is an uncertainty in the transitions of a system which can be described as a state which has a single event specified as providing transitions to at least two different resulting states; however, only one of the transitions is actually present in the system. Other examples are discussed in Section 3. Such uncertainty results in multiple models of the system which might potentially be correct. The goal is to specify conditions and algorithms which enable the identification of the correct model in a finite number of transitions despite the presence of uncontrollable actions. In particular, an algorithm provides either notification that no more models may be controllably distinguished or a finite distinguishing language which can be used to remove an incorrect model.

Section 2 describes the method used to model the plant and the relevant controllability results. Section 3 gives some examples of how a set of potentially correct models for a system might arise. Section 4 describes the concepts and techniques used to identify a correct model from a given set of models. Section 5 provides example applications of the results.

2 Description of the Model

We use the deterministic finite state machine as a model for system behavior. In what follows, only the main features of the finite state machine model related to this work are covered in a condensed manner. A more complete development related to the finite state machine model can be found in [5, 9]. More complete descriptions of the controllability and related results can be found in [13, 3, 11].

2.1 Finite State Machines and Regular Languages

A finite state machine is represented by either a four or five tuple. Specifically, if M is a finite state machine (FSM), then one writes $M = (Q, A, \delta, q_0)$ or $M = (Q, A, \delta, q_0, Q_m)$, where

$$\begin{array}{lcl} Q & = & \text{a finite set of states,} \\ A & = & \text{a finite set of transition labels or events,} \\ \delta & = & \text{the transition function, } \delta : Q \times A \to Q, \\ q_0 & = & \text{the initial state, } q_0 \in Q, \mathit{and} \\ Q_m & = & \text{the marked states, } Q_m \subseteq Q. \end{array}$$

The transition function δ is in general a partial function: $\delta(q, \sigma)!$ denotes that the transition event σ is defined from state q. The marked states signify a subset of the state set which is used to determine acceptance of a given string. A string is ***accepted*** if the machine executing the string stops in a marked state.

A finite state machine, $P = (Q, A, \delta, q_0, Q_m)$, can also be represented as a directed graph $M = (Q, T)$ where Q and T are the sets of nodes and arcs, respectively, or states and transitions in this instance [1, 4]. Q is the set of states in the machine and $T \subseteq Q \times A \times Q$ is the set of transitions. If $q_1, q_2 \in Q$ and $\delta(q_1, \sigma) = q_2$, then one denotes the transition by the three tuple (q_1, σ, q_2).

A^* is used to denote the set of all finite sequences of symbols from the alphabet A. A language is a set of strings of elements from an alphabet. If $u \in A^*$, then $|u|$ denotes the length of u, $u(j)$ denotes the j^{th} element of the string, and the set $pr(u)$ denotes the set of all strings which are prefixes of u, i.e. for $u \in A^*$

$$pr(u) = \{s \in A^* | s = u(1) \dots u(k), 0 < k \leq |u|\}.$$

The notation $s \leq u$ is used to denote that s is a prefix of u. Note that the empty string, ε, is the length zero prefix of all strings. The concept of prefix can be extended to a language in the following manner. The ***prefix closure*** of a language L is defined by

$$\overline{L} = \{w \in A^* | \exists u \in L : w \leq u\}.$$

We use the notation $s \in ppr(u)$ or $s < u$ to signify that s is a proper prefix of u, i.e. that $s \leq u$ and $s \neq u$. This concept is extended to a language, $L \subseteq A^*$, in the following manner:

$$ppr(L) = \{s \in A^* | \exists u \in L : s < u\}.$$

For this work, we restrict our attention to the class of regular languages which is a strict subset of the class of formal languages. A basic result relates regular languages and finite state machines: a language $L \subseteq A^*$ is regular if and only if it is generated by a finite state machine [9]. Language $L_m(M)$ is the language ***marked*** or ***recognized*** by machine $M = (Q, A, \delta, q_0, Q_m)$, if

$$w \in L_m(M) \Leftrightarrow \delta(q_0, w) \in Q_m,$$

where δ is extended in the usual manner, $\delta : Q \times A^* \to Q$. A language $L(M)$ is the language ***generated*** by machine M if

$$w \in L(M) \Leftrightarrow \delta(q_0, w)!.$$

The product machine is a single machine which can be used to represent the synchronous behavior of two original machines. If machines $M_1 = (Q_1, A, \delta_1, q_{1,0}, Q_{1,m})$ and $M_2 = (Q_2, A, \delta_2, q_{2,0}, Q_{2,m})$ have the same event set, A, then the product of the two machines is denoted

$$M_1 \| M_2 = (Z, A, \delta_{\|}, z_0, Z_m)$$

where,

$$Z = Q_1 \times Q_2 \text{ and } z_0 = (q_{1,0}, q_{2,0}),$$

$$\delta_{||}((q_1, q_2), \sigma) = \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \text{if defined} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

and,

$$Z_m = Q_{1,m} \times Q_{2,m}.$$

The languages generated and marked by the product machine have a specific relation to the languages of the machines from which they are composed. If $M_{||} = M_1 || M_2$ then

$$L(M_{||}) = L(M_1) \cap L(M_2) \text{ and } L_m(M_{||}) = L_m(M_1) \cap L_m(M_2).$$

2.2 Control of Discrete Event Systems

The event set, A, can be partitioned into two sets: A_c and A_u representing controllable and uncontrollable events, respectively. A language K is ***controllable*** with respect to language L if

$$\overline{K}.A_u \cap L \subseteq \overline{K},$$

where $a.b$ denotes concatenation and is often denoted by ab. A supervisor for a plant, modeled with finite state machine P, where $L = L(P)$, is a map

$$f : L \to 2^A$$

which specifies a set of inputs enabled by the supervisor which can be applied as a function of the string in L of events which the plant has previously executed. The closed loop system consisting of a supervisor, f, and plant, P, has the closed loop behavior denoted by L_f, and is defined as follows:

1. $\varepsilon \in L_f$,
2. $w\sigma \in L_f$ if and only if $w \in L_f, \sigma \in f(w)$, and $w\sigma \in \overline{L}$.

A supervisor, f, is ***complete*** with respect to a given plant P, with $L = L(P)$, if all uncontrolled actions of the plant are respected, i.e., if $x \in L_f$ and $x\sigma_u \in L$, then $x\sigma_u \in L_f$ where $\sigma_u \in A_u$ and L_f is the closed loop behavior as discussed above. The following result is a basic theorem relating these concepts.

Theorem 2.1 ([12]) For non-empty $K \subseteq L$, there exists a complete supervisor f such that $L_f = K$ if and only if K is prefix closed and controllable.

The region of weak attraction, as discussed in [3, 11], can be directly related to distinguishing different machines. The region of weak attraction for a specified set of states can be described informally as the set of states from which the system can be controlled so as to enter the set of specified states in a finite number of transitions.

The ***region of weak attraction***, $\Omega_M(G)$, for a machine, $M = (Q, A, \delta, q_0)$, or in graph notation, $M = (Q, T)$, and a specified subset of states, $G \subseteq Q$, can be determined by the algorithm in [3]. For a specific calculation of the region of weak attraction of a given set of states, G, the transitions used in its construction are denoted by $T_\Omega(G)$. This algorithm builds the region of weak attraction starting from G. Each iteration of the algorithm adds states to the region defined in the previous iteration. A state is added to the region of weak attraction only if there is an event σ which describes a transition into the region defined in the previous iteration and there does not exist an uncontrolled event to a state not in the region defined by the previous iterations of the algorithm. The transition labeled by this σ is added to $T_\Omega(G)$ as are the uncontrolled transitions from this state. The states in $\Omega_M(G)$ are well defined; as discussed in [3], the transitions chosen for $T_\Omega(G)$ are not necessarily uniquely defined. The algorithm is guaranteed to terminate by the finite state description of the machine. An efficient algorithm in [11] computes the region of weak attraction in $O(|Q| \cdot |A|)$ time.

The characteristics of the region of weak attraction are most easily described by certain conditions on the directed graph which describes the finite state machine. Let the machine be described by the graph $M = (Q, T)$ with $G \subseteq Q$. The region of weak attraction satisfies three main criteria as described in the following proposition.

Proposition 2.1 ([3])

$M' = (\Omega_M(G), T_\Omega(G)) \subseteq (Q, T)$

if and only if

1) M' is G-connected,

2) M' is realizable,

3) $M' - G$ is acyclic.

A graph $M = (Q, T)$ is ***F-connected*** if $F \subseteq Q$ and from every state in Q there exists a path to a state in F. A subgraph $M' = (Q', T') \subseteq (Q, T)$ is ***realizable*** if

$$(((q_1, \sigma, q_2) \in T) \wedge (q_1 \in Q') \wedge (\sigma \in A_u)) \Rightarrow ((q_1, \sigma, q_2) \in T').$$

A realizable subgraph includes all uncontrollable arcs which are defined from any state in the state set of the subgraph.

If the initial state, q_0, for the machine, M, is in the region of weak attraction, i.e., $q_0 \in \Omega_M(G)$, then we also define a machine based on the region of weak attraction, $M_\Omega(G)$, where $M_\Omega(G)$ is formally defined by the tuple:

$$M_\Omega(G) = (Q, A, \delta_\Omega, q_0, G).$$

And $\delta_\Omega(q_1, \sigma) = q_2$ if and only if (q_1, σ, q_2) is an arc defined in the construction of the region of weak attraction, i.e. $(q_1, \sigma, q_2) \in T_\Omega(G)$. With M_Ω defined, the language recognized by the resulting machine is denoted by $L(M_\Omega(G))$. As mentioned above and in [3], $M_\Omega(G)$, and hence $L(M_\Omega(G))$, is not unique.

3 Model Uncertainty

Uncertainties in the plant model provide a set of models which are potentially correct models of the plant. Each model in this set is obtained by assuming that the uncertainty results from a specific lack of knowledge about the structure of the plant.

Example 3.1 Consider an automatic guided vehicle system guided by wires in the floor of a manufacturing facility. The model of the guidance system may contain errors. Each error will produce an uncertain model of the correct system. For instance, two branch nodes in the wiring may be combined into a single node in the model, an extra branch may be in the model which is not installed in the plant, or the model may be lacking a branch which is installed in the plant. Each of these errors generates an uncertain model which can be used to define a set of potentially correct models.

□

Example 3.2 Model (A) in Figure 1 gives an example of a system with uncertainty in the transitions. For this transition uncertainty, there is a single state, q_0, in the model which has "b" transitions defined to k different states, $q_1, \ldots, q_k$; yet, in the actual system, only one of these "b" transitions is defined.

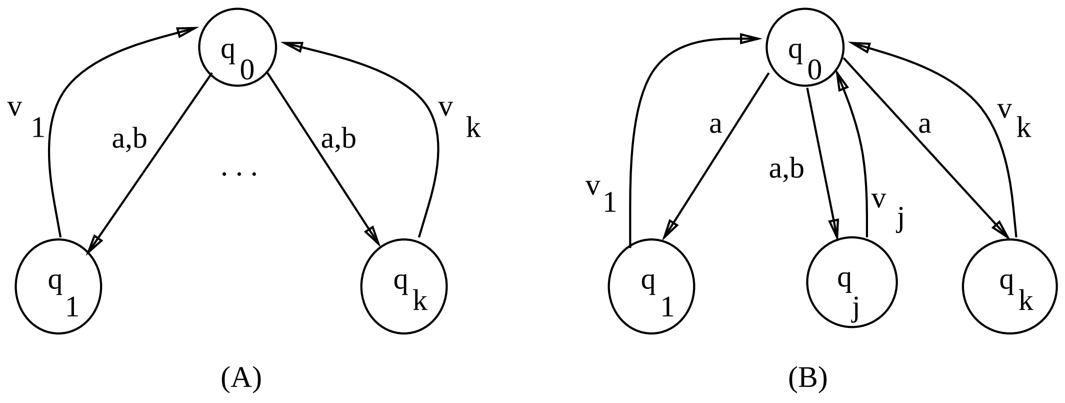

Figure 1: Model for Transition Uncertainty. (A) Model with transition uncertainty, (B) One of the possibly correct models.

□

Example 3.3 Assume that the set of events which a system can accomplish is known and that there is a known upper bound on the size of the state space. Using these two assumptions, one can construct all possible models for the system. After all unique models have been constructed, a technique is required to generate tests which can distinguish the correct model.

□

4 Distinguishing Between Models

We present deterministic techniques which provide an easily checked condition and an algorithm for correctly removing inconsistent models from consideration and identifying the correct model.

Certain concepts will provide a unified framework for the development which follows. For the following definitions, we assume that there are models $M = (Q, A, \delta, q_0)$, $M_1 = (Q_1, A, \delta_1, q_{1,0})$, $M_2 = (Q_2, A, \delta_2, q_{2,0})$, which have states $q_1 \in Q_1$, $q_2 \in Q_2$ and an event $\sigma \in A$.

Definition 4.1 Given M_1 and M_2, the predicate ***different***(z, w) holds if

$$(\delta_1(q_1, w)! \wedge \neg\delta_2(q_2, w)!) \vee (\neg\delta_1(q_1, w)! \wedge \delta_2(q_2, w)!)$$

where

$$z = (q_1, q_2), \text{ and } w \in A^*.$$

□

This predicate depends on both the state in the product machine and the string chosen to differentiate the states which make up the product state.

Example 4.1 For the machines defined in Example 3.2, let M_i and M_j be the possible models which have the b transition defined to states q_i and q_j, respectively. ***different***$((q_{i0}, q_{j0}), bv_j)$ holds, whereas, ***different***$((q_{i0}, q_{j0}), b)$ does not hold, where (q_{i0}, q_{j0}) is the product initial state.

□

In the following $A \triangle B$ denotes the symmetric difference of the two sets A and B, i.e. $A \triangle B = (A \cup B) \setminus (A \cap B)$.

Definition 4.2 A string w is a ***distinguishing string*** for languages L_1 and L_2 if

$$(w \in L_1 \triangle L_2).$$

□

Definition 4.3 A string w is a ***minimally distinguishing string*** for languages L_1 and L_2 if

$$(w \in L_1 \triangle L_2) \wedge (ppr(w) \subseteq L_1 \cap L_2).$$

□

A minimally distinguishing string is minimal in the sense that no prefix is also a distinguishing string.

Definition 4.4 A non-empty language L is a ***distinguishing language*** for L_1 and L_2 if $w \in L$ implies that w is a minimally distinguishing string for L_1 and L_2.

$\Box$

Hence, a string in a distinguishing language is one which uses the last event to distinguish between L_1 and L_2. Note that, in general, there is not a unique distinguishing language for two machines M_1 and M_2.

Example 4.2 Let $L_1 = a^*$ and $L_2 = a^*b^*$ be two languages which describe the behavior of two possible models of a black box machine. For these languages, $L_1 \cup L_2 = a^*b^*$ and $L_1 \cap L_2 = a^*$; consequently, $L_1 \triangle L_2 = a^*b^*b$. For a distinguishing language L to satisfy $(ppr(L) \subseteq L_1 \cap L_2)$, we must have that $L \subseteq a^*b$.

Observe that if the machine executes the final b, then L_2 is the correct language, and if not, then L_1 is the correct language.

$\Box$

For languages generated by a state machine, we have the following result.

Proposition 4.1

Let L_1 and L_2 be languages generated by the machines M_1 and M_2.

There exists a distinguishing language, L, for L_1 and L_2

if and only if

$(\exists w \in A^* : \textbf{\textit{different}}(z_0, w))$, where $z_0 = (q_{1,0}, q_{2,0})$.

Proof:

$\Longrightarrow$

Let $w \in L$. Since $w \in L_1 \triangle L_2$, we immediately have that $\textbf{\textit{different}}(z_0, w)$ holds.

$\Longleftarrow$

Let w be the string which satisfies $\textbf{\textit{different}}(z_0, w)$. From the definition of $\textbf{\textit{different}}(\cdot, \cdot)$, there is some $v \in L_1 \cap L_2$ such that $v < w$ and a $\sigma \in A$ such that $v\sigma \notin L_1 \cap L_2$ and $v\sigma \leq w$. Set $L = \{v\sigma\}$. It is clear that L is a distinguishing language for L_1 and L_2.

$\Box$ End proof of Proposition 4.1.

Example 4.3 For the machines in Example 4.1, ***different***$((q_{i0}, q_{j0}), bv_i)$ holds; consequently, there is a distinguishing language, $L = bv_i + bv_j$, where $a + b = \{a, b\}$.

□

4.1 Distinguishing Between Two Models

Assume that machine M has an uncertainty which causes the set of potentially correct models to consist of the models M_1 and M_2. We assume that one of these models is correct. The models are specified by the following tuples:

$$M_1 = (Q_1, A, \delta_1, q_{1,0}), \text{ and } M_2 = (Q_2, A, \delta_2, q_{2,0}).$$

The languages generated by these models are referenced by $L(M_1)$ and $L(M_2)$. We also refer to the standard synchronous product machine:

$$M_{||} = M_1 || M_2 = (Z, A, \delta_{||}, z_0).$$

The set of states in the product machine which can be used to controllably distinguish the two models is defined in the following manner.

Definition 4.5 G is the ***controllably distinguishing set*** of states for M_1 and M_2 if

$$G = \Big\{ z \in Z | \ (\exists \sigma \in A : \ \textit{\textbf{different}}(z, \sigma)) \wedge (\neg \exists \sigma_u \in A_u : \delta_{||}(z, \sigma_u)!) \ \Big\},$$

where $M_1 || M_2 = (Z, A, \delta_{||}, z_0)$.

□

A particular event which can be used to distinguish two states is called a ***controllably distinguishing event.*** Hence, an event σ is a controllably distinguishing event for z if ***different***(z, σ) holds and there is not an uncontrolled event σ_u defined in the product machine from z.

Example 4.4 For the machines given in Figure 2, let $A_u = \{b\}$. The controllably distinguishing state set is $G = \{(q_1, q), (q_2, q)\}$. In this example, the set of controllably distinguishing states is the entire product space.

□

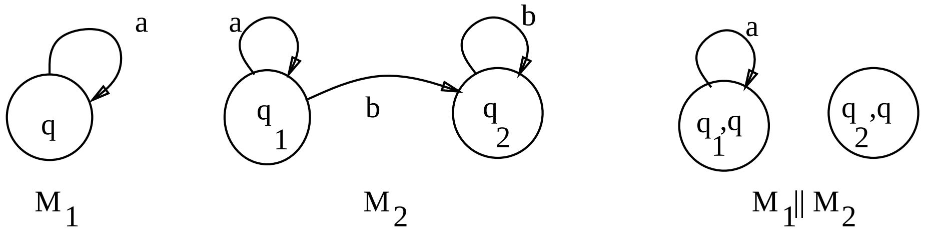

Figure 2: Machines for controllably distinguishing state set calculation.

Example 4.5 Consider the same machines as Example 4.4, but let $A_u = \{a\}$. In this instance, the set of controllably distinguishing states is $G = \{(q_2, q)\}$.

□

To controllably distinguish states in the product machine, a string must be found which leads to a state in G. Note that G is a superset of states which can be used to distinguish states in the product machine and reached from the initial state. This inclusion is a result of defining G to be all states in the product machine which have controllably distinguishing events defined without consideration of reachability constraints.

Proposition 4.2 states that there is a finite controllable method for distinguishing between two finite state machines if and only if the initial state of the product machine is in the region of weak attraction of the set of states which can be used to distinguish between the two machines.

Proposition 4.2

Let $M_1 = (Q_1, A, \delta_1, q_{1,0})$ and $M_2 = (Q_2, A, \delta_2, q_{2,0})$, be two machines.

There exists a finite language L which satisfies:

1. L is controllable with respect to $L(M_1) \cap L(M_2)$,
2. L is a distinguishing language for $L(M_1)$ and $L(M_2)$,

if and only if

$$z_0 \in \Omega_{M_1||M_2}(G),$$

where

1. $z_0 = (q_{1,0}, q_{2,0})$,
2. G is the controllably distinguishing set of states, and
3. $\Omega_{M_1\|M_2}(G)$ is the region of weak attraction of G in $M_1\|M_2$. for M_1 and M_2.

Proof:

$\Longleftarrow$:

We will demonstrate a language L which satisfies Definition 4.4 with respect to $L(M_1)$ and $L(M_2)$, and is controllable with respect to $L(M_1) \cap L(M_2)$.

Since $z_0 \in \Omega_{M_1\|M_2}(G)$, we can define a machine $M_\Omega(G)$, as discussed in Section 2.2, and let $L_m(M_\Omega)$ be the language marked by this machine with G as the marked states. Define $X \subseteq A$ such that if $t \in L_m(M_\Omega)$ then $\exists \sigma \in X$ to satisfy the definition of G, i.e. that $(\delta_{\|}(z_0, t) = z) \wedge (z \in G) \Rightarrow \neg\delta_{\|}(z, \sigma)!$. Such a symbol is defined for every string in $L_m(M_\Omega)$ by the definition of G. Further define $L \subseteq L_m(M_\Omega)X$ by $t\sigma \in L$ if $t \in L_m(M_\Omega)$, $z = \delta_{\|}(z_0, t)$, and σ is a controllably distinguishing event for z.

We must show that this language is finite, satisfies the definition for controllability and satisfies the definition of a distinguishing language.

1. L finite:

$$\begin{array}{ll} & (\Omega_{M_{\|}}(G), T_\Omega(G)) - G \text{ acyclic} \\ \Rightarrow & \qquad \{\text{property of } \Omega_M(G)\} \\ & \text{no cycles in } M_\Omega - G \\ \Rightarrow & \\ & \neg\exists s \in L_m(M_\Omega) : |s| \geq |\Omega_{M_{\|}}(G)| \\ \Rightarrow & \\ & L \text{ finite.} \end{array}$$

2. L controllable with respect to $L(M_1) \cap L(M_2)$:

   Let $(Z_\Omega, T_\Omega) = (\Omega_{M_{\|}}(G), T_\Omega(G))$ and assume that

$(t \in \overline{L}) \wedge (\sigma \in A_u) \wedge (t\sigma \in L(M_1) \cap L(M_2))$.

$$\begin{array}{ll}
 & t \in \overline{L} \wedge z_0 \in \Omega_{M_{||}}(G) \\
\Rightarrow & \qquad \{L \subseteq L_m(M_\Omega).X \text{ and definition of } L_m(M_\Omega)\} \\
 & \delta_{||}(z_0, t) \in Z_\Omega \\
\Rightarrow & \qquad \{t\sigma \in L(M_1) \cap L(M_2)\ ,\ \Omega_M(G) \text{ realizable, } z = \delta_{||}(z_0, t)\} \\
 & \exists z' \in Z_\Omega : (z, \sigma, z') \in T_\Omega \\
\Rightarrow & \qquad \{\text{definition of } L_m(M_\Omega)\} \\
 & t\sigma \in L(M_\Omega) \\
\Rightarrow & \\
 & t\sigma \in \overline{L}.
\end{array}$$

Hence L is controllable with respect to $L(M_1) \cap L(M_2)$.

3. L non-empty:

$$\begin{array}{ll}
 & z_0 \in \Omega_{M_{||}}(G) \\
\Rightarrow & \qquad \{M_\Omega \text{ is } G\text{-connected}\} \\
 & \exists v \in A^* : \delta_{||}(z_0, v) \in G \\
\Rightarrow & \qquad \{\text{definition of } G \text{ and } X\} \\
 & \exists t = v\sigma \in A^* : t \in L_m(M_\Omega).X \\
\Rightarrow & \\
 & L \neq \emptyset.
\end{array}$$

4. $ppr(L) \subseteq L(M_1) \cap L(M_2)$:

$$\begin{array}{ll}
 & L \subseteq L_m(M_\Omega).X \\
\Rightarrow & \\
 & ppr(L) = L(M_\Omega) \\
\Rightarrow & \qquad \{\text{definition of } M_\Omega \text{ and } M_{||}\} \\
 & ppr(L) \subseteq L(M_1 || M_2) \\
\Rightarrow & \qquad \{\text{property of regular languages and FSM}\} \\
 & ppr(L) \subseteq L(M_1) \cap L(M_2).
\end{array}$$

5. $L \subseteq L(M_1) \triangle L(M_2)$:

$$\begin{array}{ll} & (t \in L) \wedge (t = v\sigma) \\ \Rightarrow & \qquad \{\text{definition of } G \text{ and } L\} \\ & z = \delta_{||}(z_0, v) \in G \\ \Rightarrow & \qquad \{\text{definition of } G \text{ and } X\}, \\ & \textit{different}(z, \sigma) \\ \Rightarrow & \qquad \{\text{definition of } G\} \\ & t \in L(M_1) \triangle L(M_2). \end{array}$$

The last three items combine to show that L satisfies the conditions for being a distinguishing language.

$\Longrightarrow$:

Let (V, T) denote the graph representation of the product machine $M_1 || M_2$. Consider the subgraph, (V_0, T_0), of (V, T) obtained by running all strings in L on (V, T). Define (V_0, T_0) by

$$\begin{array}{l} V_0 = \{z \in Q_1 \times Q_2 : (\exists w \in \overline{L} : \delta_{||}(z_0, w) = z)\} \\ T_0 = \{(z_1, \sigma, z_2) \in T : (\exists w \in \overline{L} : (\delta_{||}(z_0, w) = z_1) \wedge (w\sigma \in \overline{L}))\}. \end{array}$$

Any state in the product machine reached by a string in the closure of L is included in V_0 and any transition traversed by a string in the closure of L in included in T_0. Since L is finite and $ppr(L) \subseteq L_1 \cap L_2$, it is clear that (V_0, T_0) is well defined and that a simple algorithm will generate this subgraph.

Consider the states reached by string in L/A. (Recall, L/A denotes strings in L with the last symbol removed.) The following lemma follows easily from the definition of a distinguishing language and controllability.

Lemma 4.1 With L as assumed in the statement of the proposition,

$$\forall w \in L/A : \delta_{||}(z_0, w) \in G$$

where z_0 and G are as defined in the statement of Proposition 4.2.

The following lemma provides the realizability of (V_0, T_0) with respect to (V, T) and A_u.

Lemma 4.2 (V_0, T_0) is realizable with respect to (V, T) and A_u.

Proof:
Let $z_1 = \delta_{||}(z_0, w)$ where $w \in ppr(L)$, $(z_1, \sigma, z_2) \in T$, and $\sigma \in A_u$. From these facts, we must show that $(z_1, \sigma, z_2) \in T_0$. Notice that from the construction of V_0, we have that $w \in ppr(L)$ implies that $z_1 \in V_0$.

$$\begin{array}{ll} & (z_1, \sigma, z_2) \in T \\ \Rightarrow & \\ & \delta_{||}(z_0, w\sigma)! \\ \Rightarrow & \qquad \{L \text{ controllable with respect to } L_1 \cap L_2\} \\ & w\sigma \in ppr(L) \\ \Rightarrow & \\ & (z_1, \sigma, z_2) \in T_0. \end{array}$$

□ End proof of Lemma 4.2.

By Lemma 4.1, (V_0, T_0) is G-connected. By Lemma 4.2, (V_0, T_0) is realizable. If there are no cycles in (V_0, T_0), then by Proposition 2.1 we have that $z_0 \in \Omega_{M_1||M_2}(G)$ as desired.

If there are cycles, more work is required. The idea is to disable a cycle and show that the remaining subgraph is still G-connected and realizable. Since (V_0, T_0) is a finite graph, there are only finitely many cycles; consequently, after disabling all cycles, we have a subgraph remaining which is G-connected, realizable, and acyclic. By Proposition 2.1, we have that $z_0 \in \Omega_{M_1||M_2}(G)$ as desired.

Now we must show that cycles may be removed while retaining the connectedness and realizability of the subgraph (V_0, T_0).

We start by classifying all transitions in T_0. A transition is included in class $\mathcal{C}$ if it must be included in T_0 for controllability reasons, i.e. if the subgraph would lose the realizability characterization by not having a specific transition, then that transition is included in $\mathcal{C}$. Hence,

$$(z_1, \sigma, z_2) \in \mathcal{C} \Leftrightarrow ((z_1, \sigma, z_2) \in T_0 \wedge \sigma \in A_u).$$

A transition is included in class $\mathcal{R}$ if it must be included in T_0 for reachability reasons, i.e. if a node would no longer be G-connected without the presence of a specific transition, then that transition is included in $\mathcal{R}$. Hence, $(z_1, \sigma, z_2) \in \mathcal{R}$ if and only if there does not exist a $w \in A^*$ such that there is a path labeled by w from z_1 to G in the subgraph (V_0, T_0), where $w(1) \neq \sigma$.

Assume that there is a cycle in (V_0, T_0). If there is a transition (z_1, σ, z_2) on the cycle which is not in $\mathcal{C} \cup \mathcal{R}$, then we can clearly remove this transition and retain the G-connectivity and realizability. This fact follows from the facts that any transition, (z_1, σ, z_2), not in $\mathcal{C} \cup \mathcal{R}$ is controllable and there is another path in the subgraph from z_1 to G which does not use the transition in question.

Hence, if we can remove all cycles by deleting transitions which are not in $\mathcal{C} \cup \mathcal{R}$, then we are done.

Assume that there is a cycle remaining which only has transitions in $\mathcal{C} \cup \mathcal{R}$. Let (z_1, σ, z_2) be a transition on this cycle. From the construction of the subgraph (V_0, T_0), we have that there is a string $s \in ppr(L)$ such that $\delta_{||}(z_0, s) = z_1$. The following lemma provides the crucial result.

Lemma 4.3 Let L and $\mathcal{C} \cup \mathcal{R}$ be as above.
If there is a cycle which has transitions only in $\mathcal{C} \cup \mathcal{R}$, then there are strings of arbitrary length in L.

Proof:
Let $x \in A^*$ be the labels of the transitions on this cycle, i.e. $\delta_{||}(z_1, x) = z_1$. Let $m = |x|$. Let z_i denote the state immediately preceding label $x(i)$, i.e. $(z_i, x(i), z_{i+1})$ is a transition on this cycle. Note that $z_{m+1} = z_1$.

From the controllability of L with respect to $L_1 \cap L_2$, Lemma 4.1, and the fact that every transition on the cycle is in $\mathcal{C} \cup \mathcal{R}$, it is clear that $sx(1) \in ppr(L)$. The same result holds for each i, i.e. $\forall i : 1 \leq i \leq m : sx(1) \ldots x(i) \in ppr(L)$. Hence, $sx \in ppr(L)$, implying that $sx^* \subseteq ppr(L)$.

□ End proof of Lemma 4.3.

This last statement contradicts the finiteness of L; consequently, there can be no cycle consisting only of transitions from $\mathcal{C} \cup \mathcal{R}$. As a result of Lemma 4.3, any cycle in the subgraph can be removed while retaining G-connectivity and realizability. Hence, the conditions of Proposition 2.1 are satisfied and we get that $z_0 \in \Omega_{M_1||M_2}(G)$.

□ End proof of Proposition 4.2.

Based on Proposition 4.2, we define a predicate which is true if two machines can be distinguished as described in the proposition.

Definition 4.6 Given machines M_1 and M_2, where $M_1 = (Q_1, A, \delta_1, q_{1,0})$ and $M_2 = (Q_2, A, \delta_2, q_{2,0})$, the predicate $disting(M_1, M_2)$ holds if $(q_{1,0}, q_{2,0})$

$\in \Omega_{M_1||M_2}(G_{1,2})$, where,

$$G_{1,2} = \left\{ (q_1, q_2) \in Q_1 \times Q_2 | \begin{array}{l} (\exists \sigma \in A : \textit{\textbf{different}}((q_1, q_2), \sigma)) \\ \wedge \\ (\neg \exists \sigma_u \in A_u : \delta_{||}((q_1, q_2), \sigma_u)!) \end{array} \right\}.$$

□

If two machines satisfy this predicate, then the machines, or their languages, are said to be ***controllably distinguishable***.

The following corollary provides the application of Proposition 4.2 to resolving an uncertainty which gives two potentially correct models. If $z_0 \in \Omega_{M_{||}}(G)$, then a supervisor corresponding to the machine represented by the graph created by the algorithm which generates the region of weak attraction can be built which will constrain the behavior of the unknown system such that an incorrect model can be identified in a finite number of transitions. (See [13] for details on how a machine representation of a supervisor is used to control a plant.)

Corollary 4.1 Let M_1 and M_2 be two models, such that one of them correctly models the plant, M. Let $M_{||} = M_1 || M_2$, $z_0 = (q_{1,0}, q_{2,0})$, and G be the controllably distinguishing set of states for M_1 and M_2.
$z_0 \in \Omega_{M_{||}}(G)$
if and only if
the correct model can be chosen in a finite number of transitions.

Proof:
$\Longrightarrow$:
By Proposition 4.2, if $z_0 \in \Omega_{M_{||}}(G)$, then there exists a finite non-empty controllable language, L, such that any string in the language can distinguish $L(M_1)$ and $L(M_2)$. By Theorem 2.1, a supervisor f can be constructed such that $L_f = \overline{L}$; hence, the plant can be controlled so as to execute strings from L. Since one of the models correctly models the plant, $ppr(L) \subseteq L(M)$, and any proper prefix of a string t in L can be executed by the plant. Since $L \subseteq L(M_1) \triangle L(M_2)$, the last symbol in the string will either be executed or not depending on whether $M = M_1$ or $M = M_2$ and which model has t defined in its language.
$\Longleftarrow$:

By Proposition 4.2, if $z_0 \notin \Omega_{M_{\|}}(G)$, then we can construct a string of arbitrary length which the plant can execute such that no supervisor may disable events such that an inconsistency is observed.

□ End proof of Corollary 4.1.

The complexity of this approach is governed by the necessity to consider the product machine for M_1 and M_2 in order to determine the distinguishing language. This operation requires $O(|Q_1||Q_2|)$ operations. In this paper, the dependency of the complexity on the size of the event set A is assumed to be a constant factor; hence is not included in the expression for the order of complexity. As shown in Section 5.1, this is a sharp bound on the complexity.

4.2 Distinguishing Multiple Models

The technique for distinguishing between multiple models with a reset capability available is an extension of the technique used to distinguish between two models. The strategy is to construct a product machine from two models in the set of models which results from considering all possible permutations of the uncertainties. Then, from this product machine, calculate the region of weak attraction for the set of controllably distinguishing states for these two models as described in Corollary 4.1. By using the machine generated by the region of weak attraction as a supervisor for the plant, at least one of these models can be removed from the set of possibly correct models by controlling the plant to enter a state which is a component of one of the product states in the set of controllably distinguishing states. Then, after at least one of these models has been eliminated as a possibly correct model, reset the plant and start the procedure over with another pair of models. Note that it is possible that neither of the models which are chosen is the correct model for the system; hence, the plant might generate a string which is not defined in either of the models used to generate the supervisor. In this case, both models are removed and the procedure continues by choosing another two models. This procedure continues until all uncertainties have been resolved or until no pair of models can be found which satisfy the conditions of Corollary 4.1.

We denote each possible model as: $M_i = (Q_i, A, \delta_i, q_{i,0}, Q_{i,m})$, where $i = 1, \ldots, k$, and k is the initial number of models from which the correct model is to be chosen. We denote the initial set of all possible models by

S_0. Using the notation given,

$$S_0 = \{M_i | i = 1, \ldots, k\},$$

where each model is in minimal canonical form [5, 9].

The following algorithm specifies the procedure given above. P denotes the actual machine or the plant which is to be correctly modeled.

Algorithm 4.1

```
*Input:*

    $S_0$ as given above.

*Output:*

    $S_n = \{M_i |$ no additional uncertainties can be resolved$\}$.

*Algorithm:*

    $p = 0$.
    While $(|S_p| > 1) \wedge (\exists M_i \in S_p \wedge \exists M_j \in S_p : \mathit{disting}(M_i, M_j))$:
        Calculate $\Omega_{M_i \| M_j}(G_{i,j})$.
        Use $M_\Omega(G_{i,j})$ as supervisor for $P$.
        Determine which model is still a possibly correct model and
        which model is not consistent with the plant.
        $S_{p+1} = S_p \backslash \{$ models which have been determined to be
        inconsistent with plant $\}$.
        Reset the plant.
        $p = p + 1$.
    End while.

End of algorithm.
```

In the worst case, the product for every pair of models would need to be calculated to check for pairs which satisfy $z_0 \in \Omega_{M_\|}(G_{i,j})$. Since there are k models in S, this calculation of products results in an algorithm with $O(k^2|Q|^2)$ complexity.

A slight modification of the proposed algorithm is to simulate, on all models in S_p, the strings which result from using $M_\Omega(G_{i,j})$ as a supervisor for the plant. Using this technique, any model which cannot successfully simulate the activity of the plant can be eliminated from consideration and need not be considered in any future pairing.

This modified approach also has worst case complexity of $O(k^2|Q|^2)$ since there is no guarantee that more than one model will be eliminated on each iteration. Also the actual complexity to accomplish the simulation results in an additional $O(k|Q|^2)$ term in the operation count. These counts are a result of the following reasoning. Each calculation of $\Omega_{M_\|}(G)$ adds a $|Q^2|$ term to the count. There are at most k^2 pairs which have to be calculated, hence, the $O(k^2|Q^2|)$ term in the count. To simulate any test string on all remaining potential models, $O(k|Q^2|)$ operations are necessary, hence, this term is added to the count retaining an overall complexity of $O(k^2|Q^2|)$.

To demonstrate the correctness of the algorithm, several points must be addressed: insuring that only bad models are removed from S_i, that the order of choosing models for the test $disting(M_i, M_j)$ does not affect the output, and that there is no other technique which might produce a smaller set of potential models.

4.2.1 Only Bad Models Removed

The first point is easily addressed. A model is removed if it is inconsistent with the plant. An inconsistency arises from either the plant executing a transition which is not in the model, such as an uncontrolled transition, or the plant not executing a transition which is defined in the model, such as from a state at which a single controlled or uncontrolled transition is defined in the model but is not executed by the plant. Hence, only "bad" models are removed from the set used to keep potentially correct models. Note that a consistent model will not be removed from this set.

4.2.2 Order Does Not Affect the Result

The second point is more subtle. *A priori* it appears that the order of testing models might be significant. I.e. there might be some incorrect model A which, when combined with another incorrect model B, generates a test string which provides that model B is removed, but that when model B is combined with the correct model a test string cannot be generated.

That the order does not matter follows from the following proposition. Proposition 4.3 states that if a pair of models, M_i and M_j, satisfy $z_0 \in \Omega_{M_i\|M_j}(G_{i,j})$ and if M_i is removed from the set of possibly correct models, then the correct model M_c and M_i satisfy $z_0 \in \Omega_{M_i\|M_c}(G_{i,c})$. Hence, if model M_j can be used to remove M_i, then model M_c can be used instead.

In the statements of the following propositions, the language generated by model M_i which is usually denoted by $L(M_i)$ is denoted by L_i. In the proof of Proposition 4.3, a language L is used to link the fact that the initial state is in the region of attraction of each of the product machines. The conditions on L are very similar to the conditions for a distinguishing language for M_i and M_j; however, the fact that neither M_i nor M_j might be the correct model requires that slightly different characteristics describe how M_j can be used with M_i to generate a supervisor which will cause M_i to be removed from the set of possibly correct models. In the following, M_c denotes the correct model. $z_{0,i,j}$ is the initial state of $M_i\|M_j$. $z_{0,i,c}$ is the initial state of $M_i\|M_c$. $G_{i,j}$ is the set of controllably distinguishing states for $M_i\|M_j$. $G_{i,c}$ is the set of controllably distinguishing states for $M_i\|M_c$.

Proposition 4.3 If $z_{0,i,j} \in \Omega_{M_i\|M_j}(G_{i,j})$ and M_i is removed from the set of possible machines, then $z_{0,i,c} \in \Omega_{M_i\|M_c}(G_{i,c})$.

Proof:
Consider the language L marked by the supervisor generated by $\Omega_{M_i\|M_j}(G_{i,j})$ and used to remove M_i. By the assumptions in the proposition statement, this language generates tests which are used to remove M_i from the set of potentially correct models.

We first describe some characteristics which this language satisfies.

Lemma 4.4

(a) L controllable with respect to $L_i \cap L_j \cap L_c$,

(b) $L \subseteq (L_i \triangle L_c) \cap (L_j \cup L_c)$,

(c) $ppr(L) \subseteq L_i \cap L_j \cap L_c$, and

(d) L is finite and non-empty.

Proof:
The controllability of L with respect to $L_i \cap L_j$ follows from the construction of $\Omega_{M_i||M_j}(G_{i,j})$ and the fact that $\Omega_M(G)$ is realizable. That L is controllable with respect to $L_i \cap L_j \cap L_c$, follows from the fact that all prefixes of the closed loop behavior are necessarily constrained to L_c.

The strings which occur in the distinguishing language generated by $\Omega_{M_i||M_j}(G_{i,j})$ consist of the following types:

(1) strings which occur in the plant, M_i, and M_j, i.e. $L_i \cap L_j \cap L_c$,

(2) strings which occur in the plant and M_j but not in M_i, i.e. $L_c \cap L_j \cap L_i^c$,

(3) strings which occur in M_i and M_j but not in the plant, i.e. $L_c^c \cap L_i \cap L_j$, and

(4) strings which occur in the plant but not in M_i and M_j, i.e. $L_c \cap (L_i \cup L_j)^c$,

For this supervisor to cause M_i to be removed from the set of possibly correct machines, the strings which occur in all three must be in the prefix of strings which will cause M_i to be removed. Hence, $ppr(L) \subseteq L_i \cap L_j \cap L_c$.

For this supervisor to cause M_i to be removed from the set of possibly correct machines in a controllable fashion, we claim that only the strings in (2), (3), and (4) above can occur as strings in the language. (See shaded areas in Figure 3.) No other string can occur and still allow M_i to be removed from the set of possibly correct machines. Any other string would not allow M_i to be removed.

From this observation we have that:

$$\begin{aligned} & L \subseteq (L_c \cap L_j \cap L_i^c) \cup (L_c^c \cap L_j \cap L_i) \cup (L_c \cap (L_j \cup L_i)^c) \\ \Rightarrow \quad & L \subseteq ((L_j - L_i) \cap L_c) \cup ((L_j \cap L_i) - L_c) \cup (L_c - (L_j \cup L_i)), \end{aligned}$$

which gives part b).

The finiteness of L is a result of the fact that $\Omega_{M_i||M_j}(G_{i,j})$ is acyclic. That L is non-empty is a result of the fact that M_i is removed, i.e. at least one event must be used to determine that M_i is not correct.

□ End proof of Lemma 4.4.

We now use this language to demonstrate that $z_{0,i,c} \in \Omega_{M_i||M_c}(G_{i,c})$. We demonstrate this fact by verifying that L satisfies the requirements of Proposition 4.2 for M_c and M_i.

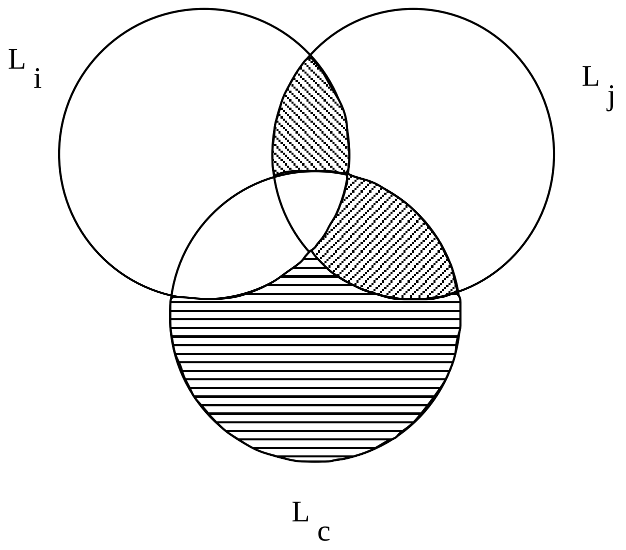

Figure 3: Parts of languages which allow removal of M_i.

1. L finite and non-empty:

   L is finite and non-empty by hypothesis.

2. L controllable w.r.t. $L_c \cap L_i$:
   From the definition of $\overline{L}$, we have that $(t \in \overline{L}) \Rightarrow t \in L \cup ppr(L)$.

$$
\begin{array}{ll}
 & (t\sigma_u \in L_c \cap L_i) \\
\Rightarrow & \\
 & t \in (L_i \cap L_c) \\
\Rightarrow & \qquad \{(b) \text{ and } (c)\} \\
 & t \in ppr(L) \\
\Rightarrow & \qquad \{(a) \text{ and } (c)\} \\
 & t\sigma_u \in \overline{L} \\
\Rightarrow & \\
 & L \text{ controllable w.r.t. } L_i \cap L_c.
\end{array}
$$

3. $L \subseteq L_c \triangle L_i$:

$$L \subseteq (L_i \triangle L_c) \cap (L_j \cup L_c)$$
$$\Leftrightarrow$$
$$L \subseteq ((L_j - L_i) \cap L_c) \cup (L_c - (L_j \cup L_i)) \cup ((L_j \cap L_i) - L_c)$$
$$\Rightarrow$$
$$L \subseteq (L_i^c \cap L_c) \cup (L_c \cap L_i^c) \cup (L_i \cap L_c^c)$$
$$\Rightarrow$$
$$L \subseteq (L_c \cap L_i^c) \cup (L_i \cap L_c^c)$$
$$\Rightarrow$$
$$L \subseteq L_c \triangle L_i.$$

4. $ppr(L) \subseteq L_c \cap L_i$:

$$ppr(L) \subseteq L_i \cap L_j \cap L_c$$
$$\Rightarrow$$
$$ppr(L) \subseteq L_c \cap L_i$$

Since L satisfies the requirements for Proposition 4.2 with respect to M_i and M_c, we immediately have that $z_{0,i,c} \in \Omega_{M_i||M_c}(G_{i,c})$.

□ End proof of Proposition 4.3.

Proposition 4.3 provides that the order does not matter when choosing which pair of models to use to generate the next test. When combined with the first point, that only "bad" models are removed, we have that it is sufficient to test bad models with the correct model, which will never be removed from the set of potentially correct models.

4.2.3 Optimal Complexity of Algorithm

Now we address the question of whether some other procedure might be used to generate a smaller set of potentially correct models.

Proposition 4.4

Algorithm 4.1 provides a minimal set of potentially correct models.

(Minimal in the sense that there does not exist another technique to controllably remove more models than are removed by Algorithm 4.1 from the set of potentially correct models in a finite number of transitions .)

Proof:
Let $\overline{S}$ be the set of potentially correct models output by Algorithm 4.1.

If $|\overline{S}| = 1$, then we are done, i.e. $\overline{S} = \{M_c\}$.

If $|\overline{S}| > 1$, then we know that since the test *disting*(M_i, M_j) is not satisfied for any pair of models in $\overline{S}$ and consequently

$$\forall i, j : M_i, M_j \in \overline{S} : z_{o,i,j} \notin \Omega_{M_i||M_j}(G_{i,j}).$$

In particular, we know that

$$z_{o,i,c} \notin \Omega_{M_i||M_c}(G_{i,c}). \tag{1}$$

Assume that there is some other technique which controllably removes $M_i \neq M_c$ from $\overline{S}$. To remove M_i, a test must cause M_i to reach a state at which an inconsistency with the plant arises. This state in M_i is a component to a state in $G_{i,c}$, otherwise by controllability constraints M_i cannot be removed by this test. By (1), we know that we can construct a behavior for M_i which never enters G. Hence, we have a contradiction and there cannot be any technique which can controllably generate a smaller set of potentially correct models.

□ End proof of Proposition 4.4.

4.2.4 Resolving Uncertainty Without Reset

To resolve uncertainty without a reset capability, a slight modification must be made to the algorithms given previously. The modification consists of updating the models still under consideration to reflect any actions which the actual plant has taken. This update is manifested by modifying the model descriptions so that the initial state has a dependence on events which have already occurred.

Hence, the old model $M = \{Q, A, \delta, q_0, Q_m\}$, is modified to $M(s) = \{Q, A, \delta, q_0(s), Q_m\}$, where s is the string which has been executed to this point. Note that only the initial state needs to have this dependence. The other components of the model do not need to be modified.

Note that for this modification, all models must be updated to determine if the new initial state after a test string has been executed is in the region of attraction for the set of distinguishing states. However, the actual region of attraction does not need to be recalculated because the states which can be attracted to the distinguishing states do not change with each test string, only the initial state changes.

The need to simulate the test strings does not increase the complexity of the algorithm. In the worst case, this algorithm could require that $O(k^2)$ regions of weak attraction be calculated to find enough test strings. Hence, this algorithm also has $O(k^2|Q|^2)$ complexity.

5 Examples

5.1 Optimality for Single Transition Uncertainty

This example demonstrates that resolving a single uncertainty has complexity at least as great as that of creating the product machine for the two potentially correct models. This complexity arises from the fact that the product machine is used to generate the set of controllably distinguishing states and hence the minimally distinguishing language. For this example, z is the event for the uncertain arcs and $A_u = \{a, c, d\}$. Figure 4 illustrates the two possible transition functions for the machine.

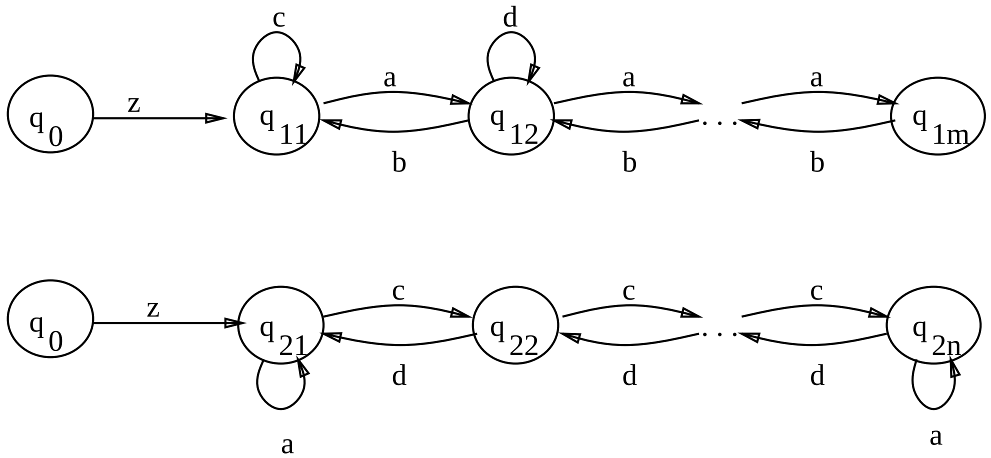

Figure 4: Set of models for which product method is optimal.

Following the procedure specified in Proposition 4.2, we create the product machine (Figure 5) and calculate the states G which can be used to distinguish q_{11} and q_{21} and the region of weak attraction for G.

From the graph representation of the transition function for the product machine, we can determine that

$$G = \{q_{1m,2n}, q_{0,11}, \ldots, q_{0,1m}, q_{0,21}, \ldots, q_{0,2n}, \}.$$

From Figure 5, we observe that the only state in the current G which can have $q_{0,0}$ in the region of weak attraction is $q_{1m,2n}$; hence, we will limit our calculations for a new set $G' = \{q_{1m,2n}\}$. Some of the iterations of the algorithm to calculate the region of weak attraction are given:

$$\begin{aligned}
&V_0 = \{q_{1m,2n}\}\\
&V_1 = V_0 \cup \{q_{1m,2(n-1)}\}\\
&.\\
&.\\
&.\\
&V_n = V_{n-1} \cup \{q_{1(m-1),21}\}\\
&.\\
&.\\
&.\\
&V_{m*n} = V_{m*n-1} \cup \{q_{0,0}\}.
\end{aligned}$$

Hence, by Proposition 4.2, since $z_0 \in \Omega_{M_{\|}}(G')$, the two states q_{11}, q_{21} are controllably distinguishable, and the uncertain arc can be resolved. Observe that $q_{1m,2n} \in G'$ and that there is a string $z(c^n ad^n a)^m$ which can occur uncontrollably before reaching $q_{1m,2n}$; hence, to resolve the uncertainty, every state which can be reached in the product machine from the initial state might be visited. This fact demonstrates that resolving this uncertain transition requires *O(mn)* operations.

To resolve the uncertainty, construct a supervisor with finite state machine representation as shown in Figure 5 and run the unknown plant and supervisor as a closed loop system. (See [13] for more detail on this procedure.) A distinguishing language for this example is $L = z(c^n ad^n a)^m a$.

5.2 Finiteness of Languages in Proposition 4.2

This example demonstrates the requirement for finiteness in Proposition 4.2.

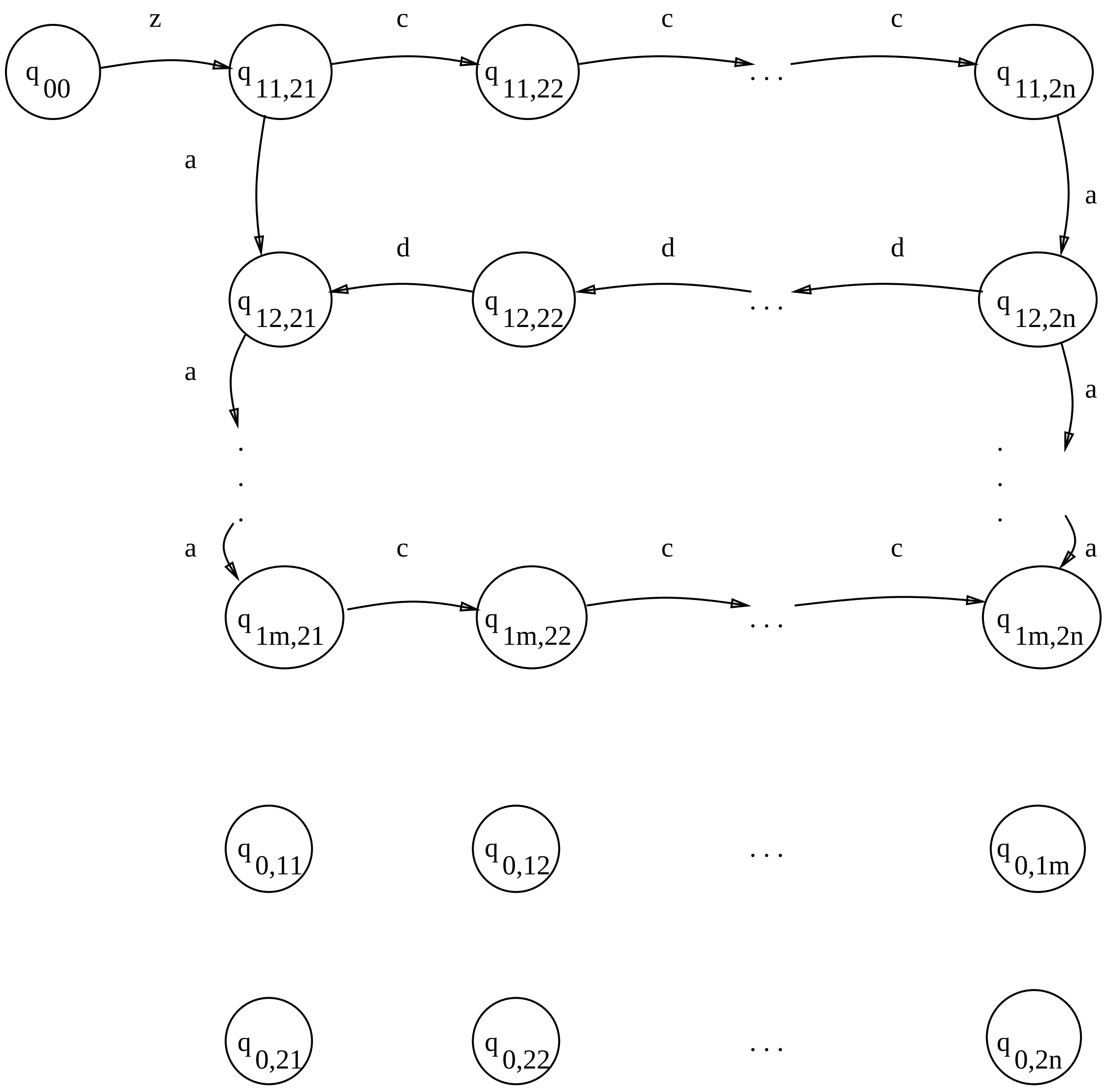

Figure 5: Product machine for system for which product method is optimal.

Let $L_1 = (av)^*$ and $L_2 = (a(u+v))^*$ be the languages for two possible models where $a \in A$ and $u, v \in A_u$. See Figure 6 for the machine representation for these languages.

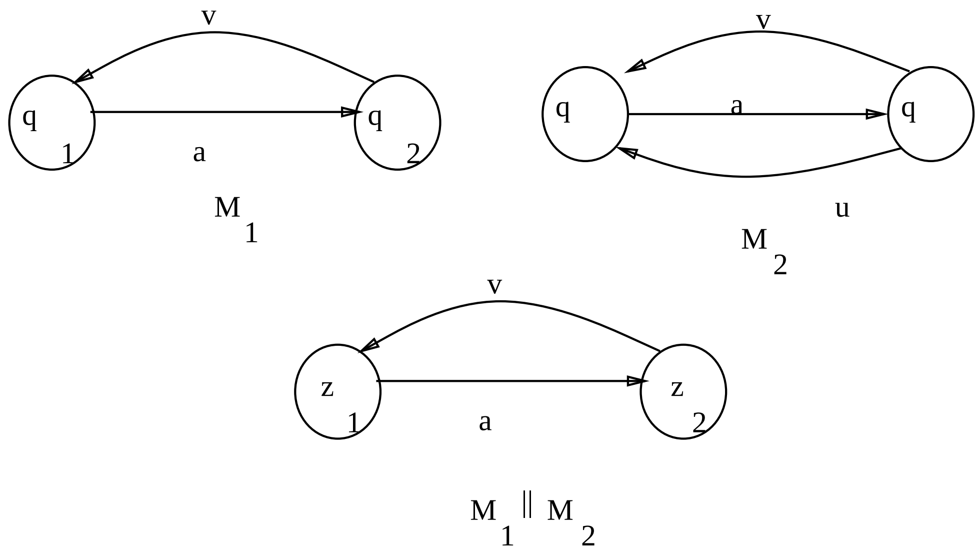

Figure 6: System to Demonstrate Requirement for Finiteness

For this example, if $L = (av)^* au$, then L is controllable with respect to $L_1 \cap L_2$, $L \subseteq L_1 \triangle L_2$, and $ppr(L) \subseteq L_1 \cap L_2$ as required for a language which can be used to distinguish L_1 and L_2 as described in Proposition 4.2; however, the initial state of the product machine is not in the region of weak attraction of the set of controllable distinguishing states, which is empty in this example.

6 Conclusions

In this paper we have presented a model of uncertainty related to the transitions of systems modeled with finite state machines. We developed a test for determining whether or not such uncertainty can be controllably resolved. The test using a region of weak attraction calculation also provides an algorithm for constructing a supervisor which can resolve the uncertainties. An example demonstrating the optimality of the deterministic approach for a single uncertainty is provided. Also an example is given which demonstrates how the controllability and finiteness requirements are both necessary for Proposition 4.2. This approach to choosing the correct model can be applied in any situation which has a set of models from which the correct one should be chosen.

Several possibilities exist for extensions to this work. One possibility is to expand the model used to describe a discrete event system to one which can describe a broader category of systems, such as a Petri net [8] or algebraic [10] models. Another direction of current interest is the influence which different uncertainty models have on the control and stabilization of systems modeled with discrete event system formalisms. This influence incorporates the effect that limiting the behavior of a system to a desired constraint language would have on correctly controlling the system and resolving any uncertainty in the model. A further extension is to consider how the addition of unobserved events affects the problem described in this work.

7 Acknowledgements

The authors would like to thank Ratnesh Kumar, Steve Marcus, Alex Tomlinson, and other reviewers for critically reading and assisting in the preparation of this paper.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. ***The Design and Analysis of Computer Algorithms.*** Addison-Wesley, Reading, MA, 1974.

[2] D. Angluin and C. H. Smith. "Inductive Inference: Theory and Methods". ***Computing Surveys***, 15(3):237–269, 1983.

[3] Y. Brave and M. Heymann. "On Stabilization of Discrete Event Processes". ***Internation Journal Control***, 51:1101–1117, 1990.

[4] N. Deo. ***Graph Theory with Applications to Engineering and Computer Sciences.*** Prentice–Hall, Englewood Cliffs, N.J., 1974.

[5] S. Eilenberg. ***Automata, Languages, and Machines Volume A.*** Academic, New York, NY, 1974.

[6] E.M. Gold. "System Identification via State Characterization". ***Automatica***, 8:621–636, 1972.

[7] Y.C. Ho. "Scanning the Issue". ***Proceedings of IEEE***, 77(1):3–6, January 1989.

[8] L.E. Holloway and B.H. Krogh. "Synthesis of feedback control logic for a class of controlled Petri nets". ***IEEE Transaction on Automatic Control***, 35(5):514–523, May 1990.

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

[10] K.M. Inan and P.P. Varaiya. "Algebras of Discrete Event Models". ***Proceedings of IEEE***, 77(1):24–38, January 1989.

[11] R. Kumar, V.K. Garg, and S.I. Marcus. Language stability and stabilizability of discrete event systems. ***SIAM Journal Control Optimization***, 31(5):1294–1320, September 1993.

[12] P.J.G. Ramadge and W.M. Wonham. "Supervisory control of a class of discrete event processes". ***SIAM Journal Control Optimization***, 25:206–230, January 1987.

[13] P.J.G. Ramadge and W.M. Wonham. "The control of discrete event systems". ***Proceedings of IEEE***, 77(1):81–98, January 1989.

[14] R.L. Rivest and R.E. Schapire. "Diversity-Based Inference of Finite Automata". In ***Proceedings*** 28^{th} ***Annual Symposium on Foundations of Computer Science***, pages 78–87, Los Angeles, CA, 1987.

[15] R.L. Rivest and R.E. Schapire. "Inference of Finite Automata Using Homing Sequences". In ***Proceedings 21st Symposium on Theory of Computing***, pages 411–420, Seattle, WA, 1989.

[16] L.G. Valiant. "A Theory of the Learnable". ***Communications of the ACM***, 27(11):1134–1142, November 1984.

[17] J.C. Willems. “Paradigms and Puzzles in the Theory of Dynamical Systems”. ***IEEE Transactions on Automatic Control***, 36(3):259–294, March 1991.

[18] S.D. Young and V.K. Garg. “Transition Uncertainty in Discrete Event Systems”. In ***Proceedings 6th IEEE International Symposium on Intelligent Control***, pages 245–250, Arlington, VA, August 1991.