Control System Synthesis Through Inductive Learning of Boolean Concepts

Michael Lemmon, Panos Antsaklis, Xiaojun Yang, and Costantino Lucisano

In control, learning is often used to identify a single controller satisfying a particular performance measure. In certain cases, however, it is desirable to identify the set of all controllers which ensure that the controlled plant satisfies a control property such as Lyapunov stability, robust stability, or robust performance. A set of procedures identifying such sets of admissible solutions can be devised using boolean concept learning algorithms. Recent years have witnessed considerable interest in this type of learning procedure in the field of computational learning. The objective of this article is to provide some examples illustrating how boolean concept learning can be used in control systems. The first example examined in this article uses concept learning to identify the set of stabilizing controllers for certain classes of linear time-invariant plants. Another example illustrates the use of concept learning in the identification of discrete event system (DES) controllers.

Introduction

Modern robust control methods provide a systematic means of optimizing controller performance in the face of bounded process uncertainty. If the resulting performance of the robust control system is unacceptable, then the only recourse is to go back and reduce modeling uncertainty. In adaptive control, one can tolerate larger uncertainties for limited classes of problem uncertainty. In high autonomy systems, modeling uncertainty can be reduced by having the system "learn" the necessary process models [1]. By incorporating past behavioral experience with a priori process models, the learning algorithm reduces process uncertainty and thereby increases the maximum performance level attainable by the system.

Much of the prior work in learning control has focused on developing algorithms which identify a *single* admissible controller for the plant [1,2]. There are problems, however, where one seeks to identify a *set* of controllers, rather than a single controller [3]. In this case, the learning procedure searches for a characterization of all controllers consistent with a boolean (TRUE/FALSE) valued functional providing a simple "acceptable" or "unacceptable" assessment of a control system's performance. As an example, consider a boolean valued functional that maps all Lyapunov stabilizing controllers onto TRUE (1) and all destabilizing controllers onto FALSE (0). This boolean functional dichotomizes the set of controllers into two disjoint sets; the set that leads to stable and the set that leads to unstable control systems. The design problem associated with this functional attempts to approximate the *set* of Lyapunov stabilizing controllers for a given plant. This set represents the *concept class* of Lyapunov stability. Because the objective of the design problem is to learn a concept (i.e. all controllers satisfying the specified control property), these learning procedures are sometimes referred to as *boolean concept learning algorithms.*

The authors are affiliated with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, Indiana, 46556. This work was supported in part by grants from the National Science Foundation (MSS92-16559) and the Electric Power Research Institute (RP8030-06).

Design problems requiring boolean concept learning are easily formulated. It may, for example, be desirable to identify all controllers of a given linear or non-linear continuous-state system that render the controlled plant Lyapunov stable [3,4,5]. A similar problem might be to identify all controllers leading to systems exhibiting robust stability or performance in some appropriate sense. These types of problems are of interest because they identify a large set of admissible controllers. Once identified, this set of admissible controllers can then be used to pick out a single controller that satisfies additional control objectives. This is not unlike what happens in the design of H^∞ controllers for linear time invariant systems; all stabilizing controllers are identified via the Youla parameterization and then a single controller is selected which minimizes the ∞-norm of a particular transfer function. Another example is found in the synthesis of supervisory controllers for discrete event systems (DES) [6]. In this case, the learning algorithm needs to identify all of the controllable behaviors (i.e., the supremal controllable sublanguage) that the DES can realize. Once again, the learning procedure identifies a set of legal behaviors, rather than a single legal behavior.

There is a rich body of results [7,8] concerning the advantages and limitations of boolean concept learning. Aside from robotic exploration [9] and generalizations [10] of concept learning, boolean concept learning has not been widely used in control. The purpose of this article is to show how concept learning provides a useful framework for solving traditional control problems. This article illustrates these ideas through two different types of control problems. The first control problem is concerned with identifying a set of stabilizing controllers. Its solution is illustrated with an example involving the stabilization of an autonomous spacecraft [5]. The second design problem examines logical DES controller synthesis as a concept learning problem [11].

The remainder of this article is organized as follows. The next section introduces the notion of a control concept. The third section discusses boolean concept learning and presents a specific algorithm for learning the concept of Lyapunov stability for linear time invariant plants. The fourth section uses the learning algorithm of section three to adaptively stabilize a spinning spacecraft. Another control concept associated with the supervision of discrete event systems is then presented in the fifth section. The sixth section summarizes the principal results and contributions of this article.

Control Concepts

This section shows how boolean concepts arise in control. In particular, use of control concepts is illustrated with the concept of Lyapunov stability.

Let K denote a set of instances. A *concept* is a boolean functional $c : K \rightarrow \{0, 1\}$ over this set of instances. The semantic meaning of this function is that instances, $k \in K$, for which $c(k) = 1$ will be said to satisfy the concept c. A *concept class*, $\mathcal{C}$, will be a set of concepts. It will be convenient to decompose a concept class into a denumerable number of subclasses, $\mathcal{C}_n$, indexed by the integer n. For example, C might consist of all boolean formulae and $\mathcal{C}_n$ might consist of all boolean formulae of length n.

A control concept arises when the concept is defined over a control system's set, K, of controller gains. The boolean functional, c, characterizing the control concept maps all gain vectors onto 1 if the systems parametrized by these gains, k, all possess the same control property. An example of such a property might be Lyapunov stability, robust stability, or robust performance. A specific example was presented in the preceding paragraph where there is a boolean functional, c, whose value $c(k) = 1$ for all k that stabilize (in the sense of Lyapunov) a given plant. In this case the concept class $C = \{c\}$ represents the concept of Lyapunov stability for that particular type of control system. Another example occurs if the boolean functional, c, takes on a value of 1 for all k having a magnitude less than unity. This concept class represents the concept of controllers with unity bounded gain vectors.

Another example of a control concept is the concept of P-stability. Consider the set of Lyapunov stable controllers for linear time invariant systems of the form

$$\dot{\bar{x}} = (A + bk^T)\bar{x} = A_k\bar{x}, \qquad (1)$$

where $A \in \Re^{n \times n}$, $b \in \Re^n$, and $k \in \Re^n$. The system to be controlled will be denoted by the ordered pair, (A, b), and the controlled system has a system matrix $A_k = A + bk^T$. Lyapunov's lemma says that this system is exponentially stable if and only if there exists a positive definite symmetric matrix P such that

$$A_k^T P + PA_k < 0. \qquad (2)$$

If we fix the matrix P, then the preceding inequality can be used to define the concept subclass, $\mathcal{C}_P$, of P-stability for the system (A, b). In particular, the boolean functional

$$c_P(k) = \begin{cases} 1 & \text{if } A_k^T P + PA_k < 0 \\ 0 & \text{otherwise} \end{cases}, \qquad (3)$$

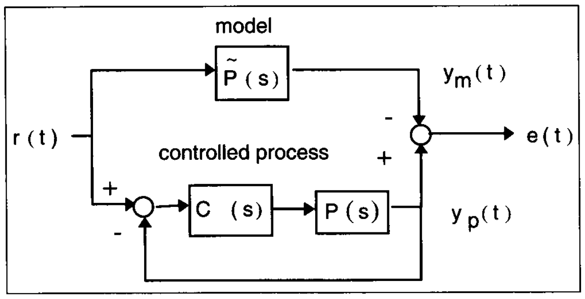

Fig. 1. Model-following control system.

denotes the concept of P-stability. The concept subclass, $\mathcal{C}_P$, consists of the single concept c_P.

The concept subclasses of P-stability can be used to characterize the concept class of Lyapunov stability, $\mathcal{C}_S$, for special types of linear systems. Assume that A_k lies in a matrix interval $[A_1, A_2]$. Recall that a matrix A lies in the matrix interval if and only if the ijth element of A is bounded above and below by the ijth elements of A_2 and A_1, respectively. From the preceding definition it is clear that the set of gains such that $A_k \in [A_1, A_2]$ forms a bounded set. Let $\mathcal{K} \subset K$ denote the set of bounded gains which stabilize the plant. It is known that a sequence of matrices, $\{P_n\}$ $(n = 1, ..., \infty)$, can be computed [12,13] to provide a necessary and sufficient condition that every system in $[A_1, A_2]$ is Lyapunov stable. This algorithm [13] provides a means of identifying stable matrix subintervals in $[A_1, A_2]$, so that every stable plant in $[A_1, A_2]$ will satisfy the Lyapunov inequality with at least one of these P-matrices. This result suggests that the concept class, $\mathcal{C}_S$, for controlled systems, A_k, in the specified matrix interval can be decomposed into a denumerable collection of concept subclasses, $\mathcal{C}_n$, associated with the concept of P_n-stabilizing controllers.

These ideas can be graphically illustrated in the following example. Consider a model following system whose block diagram is shown in Fig. 1. In this figure, the system is a linear time invariant (LTI) system. The reference plant is described by a stable minimum phase rational transfer function, $\tilde{P}(s)$. Now consider a specific model reference and plant given by $\tilde{P}(s) = 1/(s^2 + 2s + 1)$ and $P(s) = 1/(s - 2)$, respectively. Assume that the control system uses unity feedback with a precompensator whose transfer function is parametrized as $C(s \mid k_1, k_2) = k_2/(s + k_1)$. Fig. 2 shows a contour plot of the controlled error system's ∞-norm as a function of the controller parameters k_1 and k_2.

For the example system shown in Fig. 1, the concept class of Lyapunov stability, $\mathcal{C}_S$, consists of the boolean functional

$$c_S(k_1, k_2) = \begin{cases} 1 & \text{if } k_2 > 2 \text{ and } k_1 < 2k_2 \\ 0 & \text{otherwise} \end{cases} \qquad (4)$$

The region identified by c_S is easily determined by application of the Routh-Hurwitz procedure. The set of gains associated with the concept are shown in Fig. 2. It is also possible to graphically illustrate the concept subclass of P-stability. Consider the following P matrix,

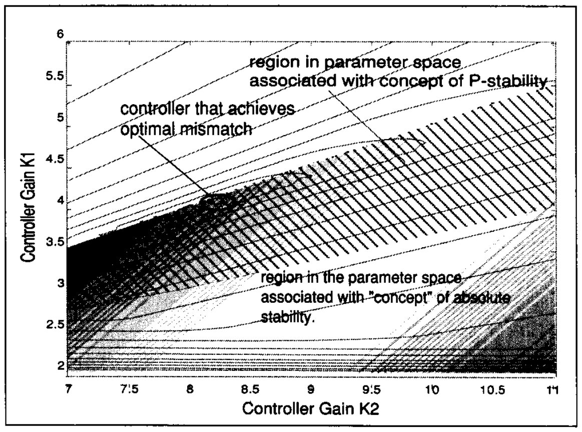

Fig. 2. Contour plot of an error system's supremum norm. This figure also shows the regions associated with the concept class of Lyapunov stability and the subclass of P-stability.

$$P = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}. \tag{5}$$

The subclass of P-stabilizing controllers, $\mathcal{C}_P$, can be shown to consist of the single concept,

$$c_P(k_1,k_2) = \begin{cases} 1 & \text{if } 25k_1^2 - 16k_1 - 20k_1k_2 + 4k_2^2 + 4k_2 + 16 < 0 \text{ and } k_2 > 5.6 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

This region is also illustrated in Fig. 2 and is seen to be a subset of the region associated with concept class $\mathcal{C}_S$.

Concept Learning in Control

The preceding section identified two control concepts for linear time-invariant control systems, Lyapunov stability, and P-stability. This section discusses methods for learning estimates of these control concepts. Concept learning methods are useful in certain control system synthesis problems. Recall that a standard approach to the synthesis of modern robust control systems is to first characterize a set of stabilizing control systems and then to select one of these admissible stable control systems to minimize an assumed performance measure. In these design procedures, it may therefore be useful to estimate the set of stabilizing control systems prior to selecting the final controller design. This section shows how concept learning methods can provide on-line identification of the concept classes $\mathcal{C}_S$ and $\mathcal{C}_P$.

Concept Learning: Preliminaries

Let $\mathcal{C}$ be a concept class consisting of boolean functional $c : K \rightarrow \{0, 1\}$. Let s be a design set consisting of N ordered pairs $(k_i, c(k_i))$ representing input-output examples of concept c. Let **S** denote the collection of all possible design sets. For example, if the concept class is $\mathcal{C}_S$ (Lyapunov stability), then each example, $(k_i, c_S(k_i))$, in the design set represents a specific controller, k_i, and the assessment, $c_S(k_i)$, of whether or not the controller stabilizes the system. In formulating the learning algorithm, it will be convenient to define another concept class, $\mathcal{H}$, consisting of boolean functionals $h : K \rightarrow \{0, 1\}$ called *hypotheses*. The class, $\mathcal{H}$, will be called the hypothesis space and a boolean functional in $\mathcal{H}$ will be called an hypothesis.

Learning algorithms can be viewed as empirical function approximation procedures. This viewpoint of learning has been discussed previously in [14]. The *function approximation problem* associated with concept learning is easily stated. Let c represent the target concept to be learned and let h represent an approximating concept drawn from the concept class of hypotheses, $\mathcal{H}$. The function approximation problem seeks to find an h which minimizes an assumed performance measure. In traditional function approximation, these measures are operator norms, but in concept learning it is convenient to use a probabilistic measure. Let $\mathbf{h} \subset K$ denote a set of gains which the hypothesis h maps onto 1 (TRUE). If an instance (controller), k, is randomly selected from $\mathbf{h}$ in a uniform manner, then the error of hypothesis h with respect to concept class $\mathcal{C} = \{c\}$ is the probability that $c(k) \neq 1$. This error is denoted as follows:

$$\text{er}(h,\mathcal{C}) = \text{Probability } \{\text{h}(k) = 1 \text{ and } c(k) \neq 1\}. \tag{7}$$

Associated with this error measure is the following function approximation problem:

$$\min_{h \in \mathcal{H}} \text{er}(h,\mathcal{C}). \tag{8}$$

Learning problems arise when the function approximation problem must be solved using a set consisting of a finite number of input/output examples of the concept. Because of its reliance on examples, this type of learning is sometimes called "empirical" function approximation. In particular, let the design set, **s**, consist of N ordered pairs $(k_i, \text{c}(k_i))$ $(i = 1, ..., N)$ where $k_i \in K$. The learning problem attempts to minimize an empirical measure of the hypothesis' error. Note that since our original error measure was a probability over the set **h**, then a useful empirical error measure is the relative frequency or ensemble average over the design set. Provided the samples in the design set are chosen in an independent and identically distributed manner, then the strong law of large numbers ensures that the sample average converges to the probability of the event. These remarks suggest the following empirical measure of hypothesis error:

$$\hat{er}(h,\mathcal{C},\text{s}) = \frac{1}{N}\sum_{i=1}^{N}\left|c(k_i) - h(k_i)\right|, \tag{9}$$

where $(k_i, c(k_i)) \in$ **s**. The learning problem is therefore concerned with the following optimization problem:

$$\min_{h \in \mathcal{H}} \hat{\text{er}}\,(h,\mathcal{C},\mathbf{s}). \tag{10}$$

A concept learning algorithm is a computational procedure which solves the preceding minimization problem. In the computational learning community, it is often more convenient to view this algorithm as a mapping $L : \mathbf{S} \rightarrow \mathcal{H}$ taking the design set onto a specific hypothesis h in $\mathcal{H}$. The learning algorithm will be

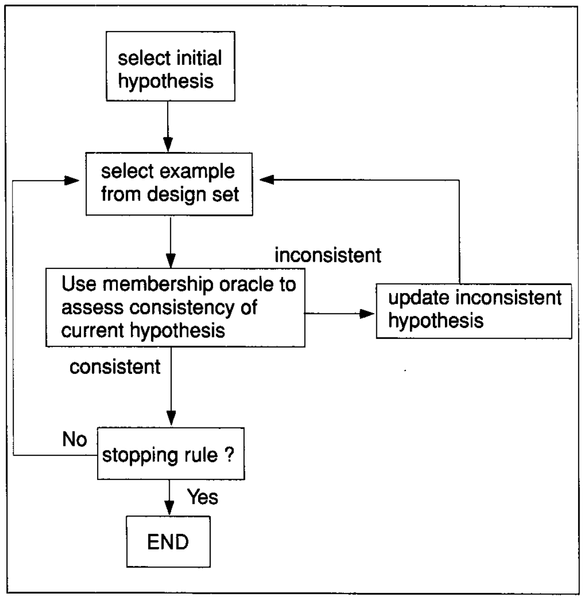
select initial hypothesis
select example from design set
inconsistent
Use membership oracle to assess consistency of current hypothesis
update inconsistent hypothesis
consistent
No
stopping rule ?
Yes
END

Fig. 3. Boolean concept learning algorithm (stability).

said to be consistent if $h(k_i) = c(k_i)$ for all gains, k_i, in the design set. Essentially a consistent algorithm produces an hypothesis whose empirical error is zero. There may, of course, be several consistent hypotheses for the given data set. An important issue in learning theory concerns the error of the hypothesis on samples not in the design set. This issue is sometimes referred to as the generalization issue. Let $h^* = L(\mathbf{s})$ be the hypothesis generated by learning algorithm L using design set $\mathbf{s}$. The learned hypothesis, h^*, will be said to generalize "well" over the concept class if the true error, $\text{er}(h^*,\mathcal{C})$, is kept small. In general, this issue is addressed by carefully selecting the concept class $\mathcal{H}$.

While the learning algorithm may be a mapping between S and $\mathcal{H}$, it is generally implemented by a computational procedure. Fig. 3 illustrates one such algorithm. The first step in this procedure selects an initial hypothesis, h, and then selects a single instance, k, from the design set $\mathbf{s}$. The instance (in our case a controller gain vector) is then evaluated by an algorithm called the membership oracle. In particular, the membership oracle evaluates $h(k)$ and $c(k)$ and declares whether they are equal or not. In the event that they are not equal, then the hypothesis h is not consistent with the design set. The learning algorithm therefore modifies the hypothesis to force consistency of h with the selected instance, k. The algorithm responsible for modifying the hypothesis is called the update procedure. A good learning algorithm eventually finds an hypothesis which is consistent with every instance in the data set. The stopping rule shown in Fig. 3 is used to stop the learning procedure. It is therefore seen that the concept learning algorithm consists of four basic components, the initialization of the hypothesis, the membership oracle, the update procedure, and the stopping rule.

The preceding generalization issue is concerned with the performance of the learning algorithm for a design set of fixed size. By the strong law of large numbers, the empirical error approaches the true error as the size of the design set increases. Therefore, if we have a consistent algorithm, we need only increase the design set's size to reduce the error of the hypothesis to an arbitrarily small level. In this framework, we then need to be concerned about the size of the design set. In particular, a learning algorithm is said to be *efficient* if the size of the design set required to produce a specified hypothesis error grows in a polynomial manner with the specified error and the learning problem's size. This notion of polynomial growth and learning efficiency has been formalized [15] under the name of *probably almost correct* (PAC) learning. In the PAC learning framework, we assume that a design set, $\mathbf{s}$, of size N has been selected in a random manner. A learning algorithm, L, is said to be PAC if there exists positive ϵ and δ such that for all $c \in \mathcal{C}$ and for all $N > N_O(\epsilon,\delta)$,

$$\text{Probability}\ \{\mathbf{s} \in S \mid \text{er}(L(\mathbf{s}),\mathcal{C}) < \epsilon\} > 1 - \delta, \qquad (11)$$

where the probabilities are taken over all possible design sets. The function $N_O(\epsilon,\delta)$ represents the size of the design set required to achieve the (ϵ,δ) accuracy specified in Equation (11). In particular, the algorithm is said to be efficient or strongly learnable if N_O is a polynomial function of δ and ϵ. Weaker notions of learnability have been introduced which only require that N_O be polynomial in δ (for fixed ϵ). It has been shown that any weak PAC learning algorithm can be modified into a strong learning algorithm [9].

Remark: Concept learning is generally studied under the name of *computational learning theory* [7]. These algorithms have also been called inductive inference procedures [16]. Prior research has identified certain classes of learnable boolean formulae (concepts) and formal languages. It is known, for example, that all conjunctive normal form expressions with k terms (k-CNF formulae) are learnable by positive examples [15]. On the other hand, boolean formulae in which each variable appears at most once (μ-formula) are not learnable [17]. In the case of formal languages, it is known that regular languages are not learnable from positive examples [18,19]. When augmented with counter-examples, however, such languages can then be shown to be learnable [20].

Learning P-Stable Concepts

This section formulates a learning algorithm for the concept of Lyapunov stability in linear time invariant plants. In particular, the objective is to formulate a concept learning algorithm to approximate the concept class of Lyapunov stability for LTI plants whose system matrix lie in a specified matrix interval. The learning algorithm to be developed has the basic structure shown in Fig. 3. The following items need to be specified in formulating the learning procedure:

- the concept class to be learned, $\mathcal{C}$
- a space of hypotheses, $\mathcal{H}$
- a set of design samples, $\mathbf{s}$
- a method for choosing an initial hypothesis
- a membership oracle
- an update algorithm
- and a stopping rule.

The concept class, $\mathcal{C}_S$, was introduced in the preceding section. In particular, it is assumed that the process to be stabilized is a linear time-invariant system with state equation 1. The gain vector, k, parametrizes the controller. It will be assumed that the system matrix $A_k = A + bk^T$ lies in a matrix interval $[A_1, A_2]$. As noted in section two, it is possible to compute a sequence, $\{P_n\}$ ($n = 1, ..., \infty$), of positive definite matrices characterizing the concept class of Lyapunov stability. The decomposition method is based on a procedure discussed in [13] and individual P matrices in the sequence are obtained by solving the linear program suggested in [12]. It is therefore possible to express the concept class of Lyapunov stability for A_k as follows:

$$C_s = \bigcup_n C_n . \tag{12}$$

If the matrix interval is small enough, then there may be a finite number of terms in Equation (12). To illustrate the following algorithm more easily, let's consider the case where there is only a single P-matrix needed to characterize C_s.

The concept class of hypotheses, $\mathcal{H}$, will be chosen to consist of boolean functionals taking ellipsoidal subsets of the gain space, K, onto 1. An ellipsoidal set, E, is characterized by the following equation:

$$E(Q, l) = \{k \in K : (k - l)^T Q (k - l) < 1\}, \tag{13}$$

where $l \in K$ is the ellipsoidal set's center and $Q \in \Re^{n \times n}$ is a symmetric positive definite matrix characterizing the ellipsoid's shape. An hypothesis, $h_{(Q,l)}(k)$, is then a boolean functional of the following form:

$$h_{(Q,l)}(k) = \begin{cases} 1 & \text{if } k \in E(Q,l) \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Denote the ith hypothesis generated by the algorithm as $h_i(k)$ and let $E(H_i, k_i)$ be its associated ellipsoidal set. Recall that the set of stable gains for A_k is denoted by $\mathcal{K}$ and forms a convex bounded subset of K. Let $\mathcal{K}$ denote this bounded set of gains. The initial hypothesis, h_0, will be chosen so that its associated ellipsoidal set properly contains $\mathcal{K}$.

Let s be a design set of finite size consisting of the controller gains which the algorithm is to learn from. In particular, if the learning procedure generates a sequence of hypothesis, h_i, then the ith example of the design set will be the center, k_i, of the ellipsoidal set associated with hypothesis h_i

The membership oracle, $m : K \rightarrow \{0, 1\}$ is a boolean functional mapping a given control gain, k, onto 1 if $c(k) \neq 1$ and $h(k) = 1$. On a semantic level, the membership oracle declares whether or not $c(k)$ renders the controlled system P-stable. Recall that we've restricted our attention to systems whose Lyapunov stability can be characterized by a single P matrix. As noted below this assumption is somewhat restrictive, but can be extended through the use of multiple P-matrices. In light of the preceding assumptions, we can evaluate $c(k)$ in the following manner. Assume that the system's current state x and state rate of change, $\dot{x}$, can be measured. Since $x^T Px$ is a global Lyapunov functional for the system, we can conclude that the inequality

$$\dot{x}^T Px > 0, \tag{15}$$

is a sufficient condition for detecting system instability. Inequality (15) can therefore be used to formulate a membership oracle (this could alternatively be called the instability-oracle) of the form

$$m(k) = \begin{cases} 1 & \text{if } \dot{x}^T Px > 0 \text{ where } \dot{x} = A_k x \\ 0 & \textit{otherwise} \end{cases} . \tag{16}$$

The preceding membership oracle provides an empirical test declaring when the system is Lyapunov unstable. In particular. the oracle declares 1 (TRUE) when the system using controller gain k is not P-stable.

Remark: Note that the preceding oracle assumed that P provides a necessary and sufficient test of system stability. In general. however, we will need to have a sequence, P_n ($n = 1, ..., N$) of matrices to characterize all stable plants in the matrix interval $[A_1, A_2]$. In this case, the membership oracle would need to be modified to the following form:

$$m(k) = \begin{cases} 1 & \text{if there exists no } n \text{ such that } \dot{x}^T P_n x < 0 \text{ for control gain } k \\ 0 & \textit{otherwise} \end{cases} . \tag{17}$$

The update algorithm for our learning procedure is formulated as follows. Note that when the membership oracle declares that k does not stabilize the system, then any gain $\bar{k}$ such that

$$x^T [Pb\bar{k}^T + \bar{k}b^T P]x > x^T [Pbk^T + kb^T P]x \tag{18}$$

will not stabilize the plant either. Since Inequality (18) is linear in $\bar{k}$, this equation represents a halfspace of gains which will not include the stabilizing gains in $\mathcal{K}$. This halfspace can be characterized by the following inequality:

$$c^T (\bar{k} - k) > 0, \tag{19}$$

where $c \in K$ is an appropriately defined real vector.

Inequality (19) now serves as the basis for the update algorithm. Fig. 4 shows the concept set, $\mathcal{K}$, and the ith bounding ellipsoid, $E(Q_i, k_i)$. Since this ellipsoid is known to properly contain $\mathcal{K}$, then the halfspace of infeasible gains identified by the above inequality cuts through the center of $E(Q_i, k_i)$. This cut is shown in Fig. 4. The intersection of $E(Q_i, k_i)$ and the complemented halfspace forms a convex body which also contains $\mathcal{K}$. This new convex body can be contained within another minimal volume ellipsoid (called the Lowner-John ellipsoid), $E(Q_{i+1}, k_{i+1})$. In particular, this updated ellipsoid is computed according to the formula [21],

$$d = \frac{Q_i c}{\sqrt{c^T Q_i c}} \tag{20}$$

$$k_{i+1} = k_i - \frac{1}{m+1} d \tag{21}$$

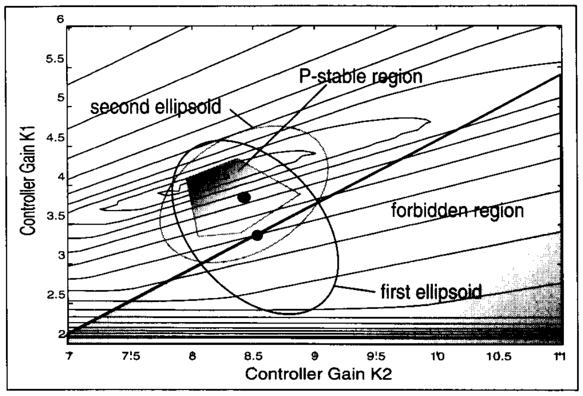

Fig. 4. Ellipsoidal update.

$$Q_{i+1} = \frac{m^2}{m^2-1}\left(Q_i - \frac{2}{m+1}dd^T\right). \tag{22}$$

The updated ellipsoid, $E(Q_{i+1}, k_{i+1})$ is shown in Fig. 4. The preceding equations constitute the update procedure used by the learning procedure.

A stopping rule can be derived using results on the finite time convergence of the ellipsoid method. As noted above, the ellipsoid method generates a sequence of ellipsoids whose volumes are monotone decreasing. It can be shown that the ratio of the volumes of any two successive ellipsoidal updates is given by

$$\frac{\operatorname{vol} E(Q_{i+1}, k_{i+1})}{\operatorname{vol} E(Q_i, k_i)} = e^{-\frac{1}{2m}}. \tag{23}$$

where m is the number of control gains to be learned. Equation (23) shows that the rate at which ellipsoid volumes decrease is independent of the iteration index, i. Therefore, it can be shown that the learning process must converge to a gain in $\mathcal{K}$ after no more than $2m \ln \frac{\operatorname{vol} E_0}{\operatorname{vol} \mathcal{K}}$ updates [21]. In other words, the proposed learning algorithm converges after a *finite* number of updates. Upper bounds on this number of updates can be easily derived [21] and provide the basis for the learning procedure's stopping rule.

Remark: Also note that the volume of concept set $\mathcal{K}$ scales as $O(m^{-m})$. Inserting this bound into the preceding convergence time shows that the learning algorithm's sample complexity is $O(m^2 \ln mL)$, where L is a measure of the desired accuracy and m is the number of controls. Therefore not only does the learning procedure converge in finite time, but this convergence time scales in a polynomial manner with problem size. This observation therefore suggests that the concept of Lyapunov stability is weakly learnable for systems whose stability can be characterized by a single positive definite P matrix.

Remark: This procedure has some obvious limitations. The first limitation is that it requires full state accessibility. For input-output systems where the states are not accessible, this approach will not be applicable.

Remark: The stability oracle requires that the plant state and its time rate of change be measured. Obviously this is not always possible. In certain cases, the state derivative can be replaced by an appropriate finite difference. There are also some applications where state derivatives can be measured directly. In systems, for example. where state information is derived from accelerometer measurements, the state derivatives are directly accessible.

Remark: The preceding discussion only pertains to interval matrices for which a single P matrix provides a necessary and sufficient condition for Lyapunov stability. Using results in [13] and [12] it may be possible to modify this algorithm to a procedure which uses multiple P-matrices to characterize the concept class, C_S. This matter is currently under investigation.

Example: Attitude Stabilization of a Satellite

In this section, the boolean concept learning algorithm outlined in the preceding section is used to stabilize a spinning spacecraft. The example is based on a recent incident with the European space agency's telecommunication satellite, Olympus [22].

The Olympus spacecraft was a telecommunications satellite built by British Aerospace for the European Space Agency (ESA). The three-axis stabilized satellite orbited about 1,000 kilometers above the Earth's surface. Because of this low orbit, there were significant disturbance torques associated with aerodynamic drag on the solar panels. There were also additional disturbance torques due to solar radiation pressure, gravity gradient effects, and bending modes. The on-board control subsystem included an analog sun sensor, three digital sun sensors, two radio frequency sensors, and two digital infrared sensors. Body rates were measured by three orthogonal gyros and a redundant skewed gyro [22]. During normal on-station operations, optical sensors were used to control the roll and pitch axes, while the yaw axis was controlled via one component of the yaw gyro pack. In all other cases, which imply the loss of Earth pointing mode, the attitude measurement was provided by gyros. Normal attitude control was achieved through the use of reaction wheels. There were also thrusters which could be used in emergency maneuvers and in off-loading of the reaction wheels.

On Aug. 11, 1993, during an abnormal Perseid meteor shower, Olympus lost Earth pointing attitude. The control mode switched automatically to a thruster-based emergency mode. This emergency maneuver, however, failed to reacquire sun pointing and introduced large body rates (spinning) as a side effect. The ground station disabled the emergency mode and the spacecraft was driven into a stable spinning state. Subsequent analysis of post-anomaly telemetry showed that there was insufficient propellant left to reestablish normal station keeping. The Olympus mission was therefore terminated a year ahead of schedule.

The Olympus scenario demonstrates the limited autonomy of existing spacecraft systems in dealing with catastrophic perturbations. The high body rates induced by the emergency maneuver spun up the spacecraft so that its original control system worked poorly. Had Olympus been able to autonomously relearn the "optimal" controller, then it may have been possible to reestablish nominal operation without depleting limited propellant reserves.

One way in which Olympus might have relearned the optimal controller gains would be to use the concept learning procedure introduced above. This learning control scheme automatically determines the state feedback gains, k, which would stabilize a linear system of the form given in Equation (1). In the context of

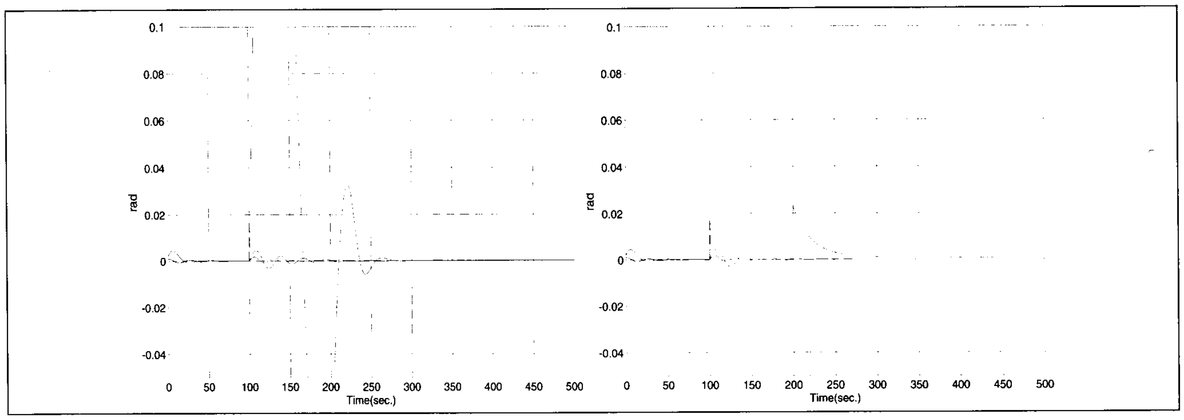

Fig. 5. Olympus simulation results. The right- and left-hand figures show the attitude angles for the conventional and learning controllers, respectively. In this case, the largest component of the spin disturbance is introduced along the roll axis (solid line). Note that the learning controller was capable of quickly removing this roll rate.

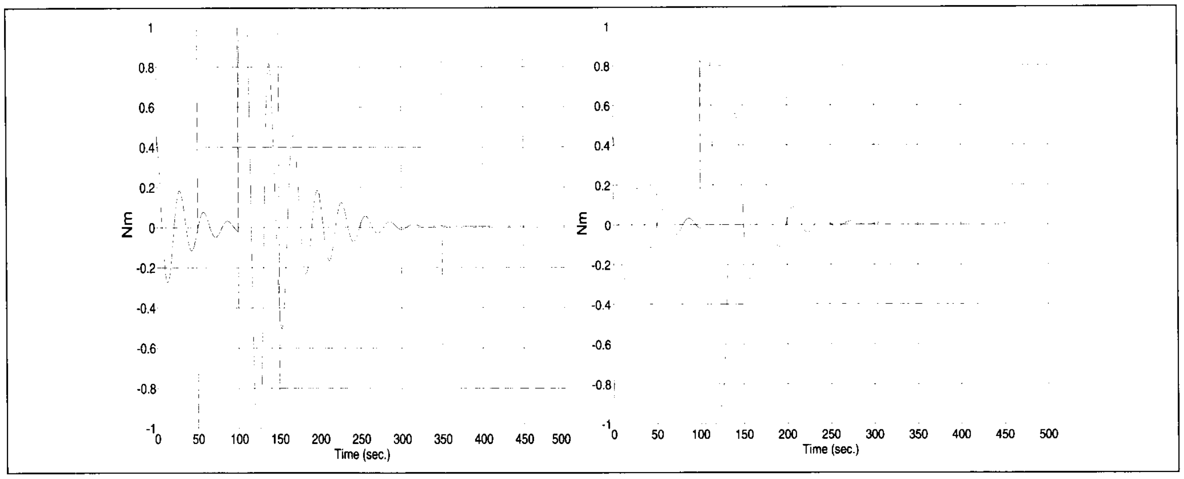

Fig. 6. Olympus simulation results. The right- and left-hand figures show the torque commands to the roll reaction wheel for the conventional and learning controller, respectively. Note that the learning controller exhibited lower peak commands with shorter duration than the conventional controller.

the Olympus scenario, these equations represent the satellite's linearized attitude dynamics. The proposed learning algorithm would therefore use state measurements to identify a gain vector k which stabilized the plant.

A concept learning procedure similar to that shown in Fig. 3 was used on the simulated Olympus scenario. An important aspect of this work was that the simulation used was a full-scale simulation of the Olympus spacecraft originally developed by British Aerospace in the design and validation of the attitude control system. The results obtained with this simulation were therefore felt to yield very accurate representations of the Olympus spacecraft's behavior. The objective was to see if using the learning algorithm would have dramatically reduced the fuel consumption during Earth reacquisition. Recall that excessive fuel consumption in recovering from the emergency maneuver was the reason for the early termination of the Olympus mission. If the learning algorithm was therefore able to significantly reduce fuel usage, the mission may not have been terminated prematurely.

Results from the simulated example are shown in Figs. 5, 6, and 7. In this example, the spin disturbance begins at t = 100 seconds. Figs. 5 and 6 show the attitude angles and reaction wheel commands, respectively, for the concept learning controller and a conventional fixed controller. Comparing the attitude angles of the two cases in Fig. 5, it is apparent that the use of the learning algorithm dramatically reduced the attitude transients experienced by the craft in recovering from the spin disturbance. Fig. 6 shows that the smaller attitude transients resulted in smaller torque commands to the reaction wheels. In fact, when the total torque commands to the roll reaction wheel were integrated, it was found that the control effort was cut in half by the application of the learning controller.

More extensive simulation experiments examined the learning controller's performance for a variety of spin disturbances. These results are shown in Fig. 7. In all cases the integrated torque

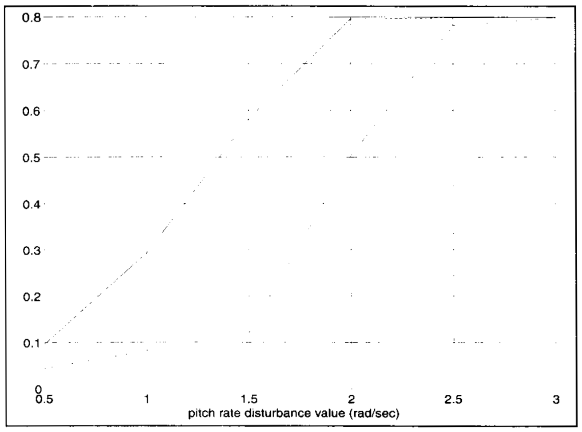

Fig. 7. Olympus simulation results. Figure plots the integrated torque command (J) vs. initial pitch rate disturbance for the conventional (solid) and learning (dashed) controllers.

commands generated by the learning controller were less than that of the fixed conventional controller. For small spin rates, the two approaches are comparable since the original controller is nearly optimal. For large spin rates, the energy savings are reduced because of the reaction wheel's limited ability to absorb excess angular momentum. The greatest energy savings occur for moderate spin rates. These savings were significant and at least on a preliminary basis it might be inferred that the use of a learning controller would have conserved sufficient fuel to prevent premature termination of the Olympus mission.

While the preceding concept learning method can be used to stabilize a continuous-time system, it can be effectively argued that there are other equally valid approaches to address this problem. The value of the above formalisms concerns the fact that we have really identified a set of controllers rather than a single one. So in reality it is the "stability concept" rather than just a stable controller that was identified. The importance of quickly identifying a set of stabilizing controllers is that after initial stabilization of the system, it is possible to select a controller from this set which is "optimal" with respect to the system's new operating conditions.

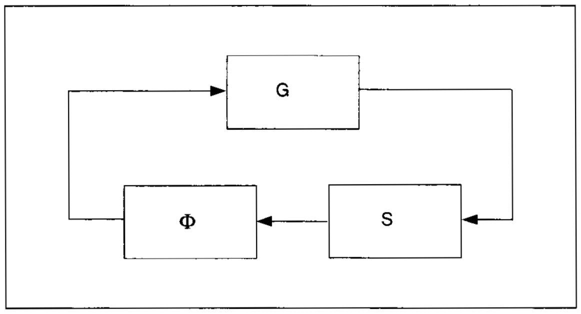

Fig. 8. DES supervisory control.

Control Concepts and DES Supervision

Discrete event systems (DES) are dynamical systems which evolve over a discrete set of symbols. In the context of control, DES are often used to model supervisory control systems. For example, consider a process control facility consisting of dozens of individual continuous-time control systems. Management of the entire plant requires the logical coordination of these various continuous-time controllers. This coordination problem can be modeled as a combination of a discrete event system (DES) plant which is being supervised by a DES controller. The DES plant generates a sequence of symbols representing significant events occurring within the plant. The controller responds to these logical events with a sequence of control directives.

There are many ways of modeling DES plants and controllers. One way models the plant and controller as a finite state machine (FSM) [6]. In particular, the plant is represented by the following FSM:

$$G = (\Sigma, Q, \delta, q_0), \tag{24}$$

where Σ represents a finite set of events that can occur in the plant, Q represents a finite set of discrete (symbolic) states. The mapping $\delta : \Sigma \quad Q \rightarrow Q$ is a transition function characterizing the evolution of the plant. There is a set of initial states q_0. The formal language generated by the given plant, denoted as $L(G)$, is well known to be a regular language. To discuss control of this plant it will be convenient to partition the plant's event set Σ into

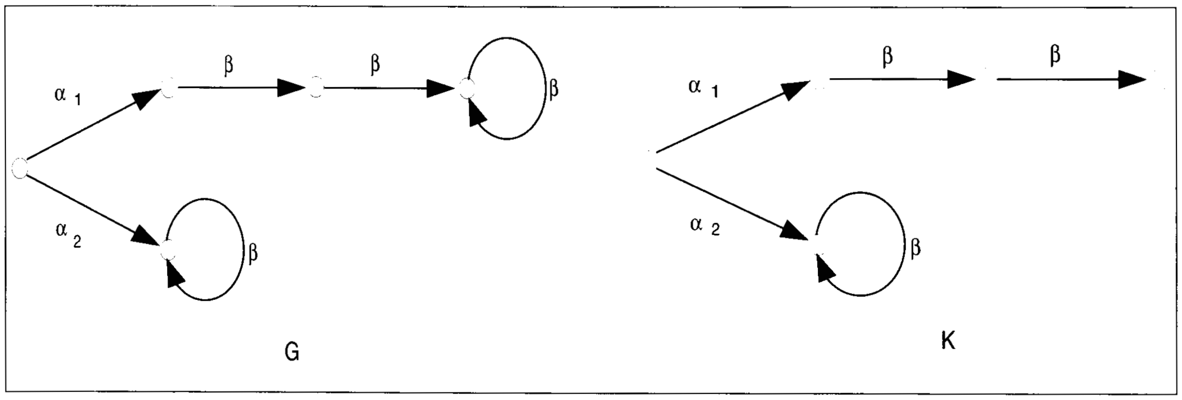

Fig. 9. The left-hand figure shows the directed graph for sample DES plant. The right-hand figure shows the directed graph for a specified behavior on this DES plant.

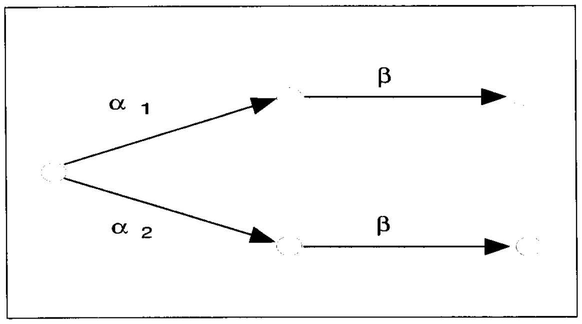

Fig. 10. L(G, 2, ε), the first lookahead window for the sample DES plant.

mutually disjoint sets of controllable, Σ_c, and uncontrollable events, Σ_{uc}. Uncontrollable events are occurrences in the plant which the supervisor cannot disable. An example of an uncontrollable event in a DES plant might be the failure of some machine in a manufacturing process. Essentially, the problem of supervisory control is to learn how to cope with uncontrollable events so that a specified behavior is realized.

The supervisor is also a FSM denoted by a 4-tuple, $S = (\Sigma, X, \xi, x_0)$. The interconnection of the plant and supervisor is shown in Fig. 8. In this figure, the supervisor receives, as input, the event traces $\sigma \in \Sigma^*$ from the plant. These received symbols drive the supervisor's state transitions. The supervisor states are symbols in the set X and the transition mapping is $\xi : \Sigma \quad X \to X$. The set of initial supervisor states is given by x_0. The supervisor generates controller directives by disabling controllable transitions in the DES plant. In Fig. 8 this is shown by a mapping, $\phi : X \to P(\Sigma)$, from the supervisor's state space onto a power set of plant events. This mapping is called the enabling function. It uses the supervisor's current state to enable a subset of plant event transitions.

The lefthand side of Fig. 9 illustrates an example of a plant FSM. The FSM is represented as a directed graph in which the nodes are states, the arcs represent the transition function, δ, and the arc labels denote plant events. In this example, the event set is $\Sigma = \{\alpha_1, \alpha_2, \beta\}$ and the uncontrollable event set, $\Sigma_u = \{\beta\}$. The regular language generated by this example can be shown to be given by strings satisfying the following formula: $L(G) = \overline{(\alpha_1\beta^2 + \alpha_2)\beta^*}$.

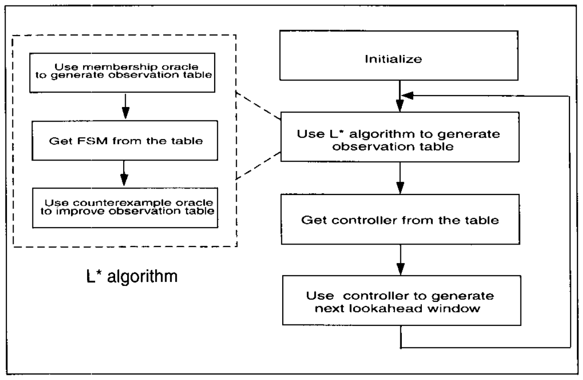
Use membership oracle to generate observation table
Get FSM from the table
Use counterexample oracle to improve observation table
L* algorithm
Initialize
Use L* algorithm to generate observation table
Get controller from the table
Use controller to generate next lookahead window

Fig. 11. On-line synthesis algorithm for DES controller.

The objective of supervisory controller synthesis is to ensure that the controlled plant realizes a specified behavior (language). In particular, the specified behavior is assumed to be expressed as a regular language, K. The righthand side of Fig. 9 shows the FSM realizing the control specification $K = \overline{\alpha_1\beta^2 + \alpha_2\beta^*}$. This specification can be also be represented as a quasi-formal set of rules. The quasi-formal control specification is itemized below:

1. If α_1 occurs first, then, at most, two β events are allowed to be generated.
2. If α_2 occurs first, then any finite β events are allowed to be generated.

Discrete event system (DES) controller synthesis methods proposed by Ramadge and Wonham [6] require that the state transition model of the desired legal behaviors be known. It has been pointed out [23,24] that this requirement is often unreasonable. In many cases, global information about the plant DES is not available, and in this case the DES plant needs to be identified from observed plant behaviors. This formulation naturally suggests that concept learning algorithms found in the computer science community might be useful in learning DES supervisors for partially specified DES plants.

Early work in on-line DES synthesis, however, suggested that inductive inference of DES supervisors would not be practical. The controllers obtained by the Ramadge-Wonham synthesis were finite automatons which serve as acceptors (FSM) for regular languages. The problem here, however, was that it had already been shown that regular sets were not learnable by design sets of positive examples [18,19]. Later work noted this fact in connection with DES controller synthesis [25]. These observations about the intractability of the inference problem for DES supervisors proved to be a considerable obstacle in the application of inductive learning methods for DES synthesis.

Modified L^* Algorithm for DES Supervision

Inference of automatons from positive examples can be viewed as a form of "passive" learning. Passive learning methods passively observe examples of the regular set and base their determination on those observations. This is precisely the type of learning procedure which was used in our spacecraft stabilization example. An alternative to "passive" observation is "active" exploration. In this case, the learning algorithm proposes specific input examples whose performance should be evaluated. Angluin showed [20] that while passive inference was NP-complete, a combination of passive and active learning was in fact polynomial in the size of the minimal FSM. The resulting L^* algorithm proposed in [20] uses passive observation and actively suggests counterexamples to assist the learning process. Subsequent work [9] extended this algorithm and used it in simple robotic navigation problems constructed on a idealized grid world.

Fig. 11 shows the flowchart for an on-line DES synthesis procedure based on the L^* algorithm. In this figure, the basic flow and components of the L^* procedure are given in the lefthand side insert. The L^* procedure, shown in Fig. 11, methodically builds up an *observation table* to represent the concept (regular set) to be learned. The table is a two-dimensional array whose rows are labeled by strings $s \in S \cup S\Sigma$ and whose columns are labeled by symbols $t \in E$. In this discussion Σ is the symbol alphabet making up the strings in the regular set, K, which we are trying to learn. Σ^* is the set of all strings obtained by concatenating symbols in Σ. S and E are prefix-closed and suffix-closed subsets of Σ^*, respectively. The table entry in the row labeled s and column labeled t will be assigned a value of 1 if the string st is legal (accepted by K). Otherwise the table entry is zero. The observation table therefore provides a very compact way of expressing the sentences of the unknown language K. The L^* algorithm is concerned with building up *complete* tables (see [20] for details) since a minimal finite state machine (FSM) consistent with the table entries can be readily written down from a completed table. The algorithm uses two types of oracles in filling out the observation table. The first oracle is associated with passive learning and is called the *membership oracle*. The second oracle accepts the generated FSM as its input and outputs a *counterexample* which is then added to the table through the use of the membership oracle. It is then conjectured that the FSM generates K and this conjecture is presented to the counterexample oracle. If the conjecture is false, the oracle returns with a *counterexample* string, which is added to the observation table through repeated queries to the membership oracle. This process then repeats until the completed finite state machine no longer generates any counterexamples.

The L^* learning algorithm cannot be directly applied to the inference of DES supervisors. This is because uncontrollable events make it difficult to define a practical membership oracle. In [11], however, some preliminary results suggested a membership oracle could be obtained by using a finite lookahead window of behaviors. The work in [11] also suggested ad hoc methods for implementing the counterexample oracle. Empirical results applying the proposed algorithm to supervisory control problems used in [6] indicate that the procedure can efficiently find the optimal DES controller. The three algorithmic components (lookahead window, membership oracle, and counterexample oracle) which were used in [11] to extend L^* learning are discussed briefly below.

- **Lookahead Window:** Rather than assuming that the plant automaton G is known, it was assumed that plant knowledge could be confined to a limited lookahead window of behaviors. This type of plant knowledge was used in [23]. The language in the lookahead window is denoted as $L(G, N, s)$. In particular, $L(G, N, s)$, denotes all strings of length no longer than N generated by the plant, G, after an observed trace s. Fig. 10 illustrates the first lookahead window, $L(G, 2, \varepsilon)$, for the plant DES (Fig. 9) as a reachability tree.
- **Membership Oracle:** The control specification K is a prefix closed regular set. Rather than knowing its finite state machine, $M(K)$, we've assumed that there exists a boolean function which declares whether or not a given string is a

T_0	ε
ε	1
α_1	1
α_2	1

$$\begin{tikzcd} q_0 \arrow[loop right, "{\alpha_1,\ \alpha_2}"] \end{tikzcd}$$

T_1	ε	α_1
ε	1	1
α_1	1	0
α_2	1	0
$\alpha_1\beta$	1	0

$$\begin{tikzcd} q_0 \arrow[rr, bend left, "\alpha_1"] \arrow[rr, bend right, "\alpha_2"'] & & q_1 \arrow[loop right, "\beta"] \end{tikzcd}$$

T_2	ε	α_1	β	$\beta\beta$
ε	1	1	0	0
α_1	1	0	1	1
$\alpha_1\beta$	1	0	1	0
$\alpha_1\beta\beta$	1	0	0	0
α_2	1	0	1	1

$$\begin{tikzcd} q_0 \arrow[r, "\alpha_1"] \arrow[r, bend right, "\alpha_2"'] & q_1 \arrow[r, "\beta"] & q_2 \arrow[r, "\beta"] & q_3 \end{tikzcd}$$

T_3	ε	α_1	β	$\beta\beta$
ε	1	1	0	0
α_2	1	0	1	1
$\alpha_2\beta$	1	0	1	1

$$\begin{tikzcd} q_0 \arrow[r, "\alpha_2"] & q_1 \arrow[loop right, "\beta"] \end{tikzcd}$$

Fig. 12. Completed observation tables and their finite state machines.

legal system behavior. This mapping is the membership oracle. No specific assumptions have been made about its implementation. In the case given above, this specification is simply given as a set of quasi-formal rules. In fact, all of our implementations of the learning procedure consistently used quasi-formal descriptions of the legal behaviors to implement the membership oracle.

- **Counterexample Oracle:** Once the L^* algorithm has constructed a completed observation table, the procedure generates counterexamples to modify the observation table. The counterexample oracle is an algorithm which searches the FSM supervisor and plant for illegal and uncontrollable behaviors. There are three ways in which a counterexample can be generated. First, we can search the conjectured supervisor's behaviors to find illegal strings accepted by the supervisor. Second, we can search the current prediction window for uncontrollable strings which can be accepted by the controller. Finally, we search the prediction window for controllable strings that are not accepted by the supervisor.

The flowchart of the on-line synthesis procedure is shown in Fig. 11. Note that since we obtain examples on-line and use that information to improve the membership oracle, the original L^* algorithm cannot be used. In the algorithm, the L^* procedure is used to generate an FSM which is as "optimal" as possible (with respect to the current specification) given the current behaviors that have been observed. Once an FSM controller has been generated it is then used on the plant process to generate new behaviors, and these new behaviors are then included into the observation table. The resulting observation table is then completed using membership queries and an associated supervisor is extracted. For the DES plant and control specification shown in Fig. 9, the proposed procedure yielded a sequence of completed observation tables and associated FSM supervisors. This sequence is shown in Fig. 12. See [11] for details about the generation of this figure. The learning procedure generated a sequence of supervisors which, in this example, terminated in a controller realizing the supremal controllable sublanguage for the example.

Summary

A standard approach in the design of robust control systems is to first identify a set of admissible controllers and then to select one that minimizes an appropriate performance measure; such is the case in H^∞ control design. Boolean concept learning provides a valuable method for implementing the first part of this approach to design. As shown in this paper, concept learning algorithms can be devised which use on-line observations to identify sets of admissible controllers. Two examples were used to illustrate these results. The first example used concept learning to identify a set of Lyapunov stabilizing controllers for a specified class of plants. The second example used concept learning to synthesize a set of legal behaviors for a discrete event system. Drawing on terminology from the computational learning theory community, these sets of admissible controllers were referred to as control concepts. The examples used in this paper illustrate the fact that for certain systems, important control concepts such as Lyapunov stability and controllability (in DES plants) are learnable concepts. The main advantage of this approach is its versatility. It appears to apply to linear, non-linear, and discrete event systems. In addition to this, since a number of control concepts can be learned in polynomial time, these algorithmic approaches represent very efficient means of learning complex controllers.

References

[1] P.J. Antsaklis and K.M. Passino, eds., *An Introduction to Intelligent and Autonomous Control*, Kluwer Academic Publishers, Boston, 1993.

[2] D.A. White and D.A. Sofge, eds., *Handbook of Intelligent Control Neural, Fuzzy, and Adaptive Approaches*, Van Nostrand, 1992.

[3] R. Kosut, M.K. Lau, and S.P. Boyd, "Set-Membership Identification of Systems With Parametric and Non-Parametric Uncertainty," *IEEE Transactions on Automatic Control*, vol. 37, pp. 929-941, 1992.

[4] M.D. Lemmon and C.J. Bett, "Direct Adaptive Stabilization of Linear Systems Using Query-Based Protocols," *Proc. of the 32nd Conference on Decision and Control*, San Antonio, Texas, Dec. 3086-3091.

[5] C. Lucisano and M.D. Lemmon, "Query-Based Attitude Control of a Low Altitude Communications Satellite," Technical Report, ISIS-94-011, Dept. of Electrical Engineering, University of Notre Dame, Notre Dame, Indiana, Sept. 1994, (also to appear in the *Proceedings of the American Control Conference*, Seattle Washington, 1995).

[6] P. Ramadge and W.M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM Journal of Control and Optimization*, vol. 25, no. 1, pp. 206-230, Jan. 1987.

[7] L.G. Valiant and M.K. Warmuth, eds., *Proceedings of the 4th Annual Workshop on Computational Learning Theory*, Santa Cruz, CA, Morgan Kaufmann publishers, 1991.

[8] M. Anthony and N. Biggs, *Computational Learning Theory*, Cambridge University Press, Cambridge, 1992.

[9] R. Schapire, *The Design and Analysis of Efficient Learning Algorithms*, MIT Press, 1992, Cambridge, MA.

[10] K.L. Buescher and P.R. Kumar, *Learning by Canonical Smooth Estimation, Simultaneous Estimation,* technical report, Dept. of Electrical Computer Engineering, University of Illinois, Urbana, IL, 1994.

[11] X. Yang, M.D. Lemmon, and P.J. Antsaklis, *Inductive Inference of Logical DES Controllers Using the L* Algorithm*, technical report, ISIS-94-010, Dept. of Electrical Eng., University of Notre Dame, Notre Dame, IN, Sept. 1994, (also to appear in the *Proceedings of the American Control Conference*, Seattle, WA, 1995).

[12] S. Boyd, Q. Yang. "Structured and Simultaneous Lyapunov Functions for System Stability Problems," *Int. J. Control*, 1989, vol. 49, no. 6, pp. 2215-2240.

[13] K. Wang, A.N. Michel, and D. Liu, "Necessary and Sufficient Conditions for the Stability of Interval Matrices," *Proceedings 32nd IEEE Conference on Decision and Control*, pp. 2014-2019, San Antonio, TX, December 1993.

[14] T. Poggio and F. Girosi, "Networks for Approximation and Learning," *Proceedings of the IEEE*, vol. 78, pp. 1481-1497, Sept. 1990.

[15] L. Valiant, "A Theory of the Learnable," *Comm. of the ACM*, vol. 27, pp. 1134-1142, 1984.

[16] D. Angluin and C.H. Smith, "Inductive Inference: Theory and Methods," *Computing Surveys*, vol. 15, pp. 237-269, Sept. 1983.

[17] M. Kearns, M. Li, L. Pitt, and L.G. Valiant, "On the Learnability of Boolean Formulae," in *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, pp. 285-295, May 1987.

[18] D. Angluin, "On the Complexity of Minimum Inference of Regular Sets," *Int. J. Information and Control*, vol. 39, pp. 337-350, 1978.

[19] E. Mark Gold, "Complexity of Automaton Identification from Given Data," *Int. J. Information and Control*, vol. 37, pp. 302-320, 1978.

[20] D. Angluin, "Learning Regular Sets from Queries and Counterexamples," *Int. J. Information and Computation*, vol. 75, no. 1, pp. 87-106, 1987.

[21] M. Groetschel, L. Lovasz, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, 1988.

[22] P. Wadey, P. Miles, and R. Lowes, "Some Aspects of the Attitude and Orbit Control Subsystem of the Olympus Spacecraft," *Proc. First ESA Internat. Conf. on Spacecraft Guidance, Navigation and Control Systems*, ESTEC, Noordwijk, The Netherlands, pp. 545-549, June 1991.

[23] Sheng-Luen Chung and Stephane Lafortune, "Limited Lookahead Policies in Supervisory Control of Discrete Event Systems," *IEEE Trans. on Automatic Control*, vol. 37, no. 12, pp. 1921-1935, December 1992.

[24] S. Young and V. Garg, "Transition Uncertainty in Discrete Event Systems," *Proceedings 6th IEEE International Symposium on Intelligent Control*, pp. 245-250, Arlington, VA, August 1991.

[25] J.N. Tsitsiklis, "On the Control of Discrete-Event Dynamical Systems," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 1, pp. 95-107, 1989.

Michael Lemmon received the B.S. degree in electrical engineering from Stanford University in 1979. He received the M.S. and Ph.D. degrees in electrical engineering from Carnegie Mellon University in 1987 and 1990, respectively. Between 1979 and 1986, he was a control systems and signal processing engineer at a number of aerospace firms, including TRW, Lockheed, and General Electric. Since 1990, he has been an assistant professor of electrical engineering at the University of Notre Dame, Notre Dame, IN. His main interests lie in the relationship between machine intelligence and control systems. He has done research in neural network learning, hybrid control systems, adaptive control systems, discrete event systems, and non-convex optimization. Lemmon served as an associate editor of the IEEE Transactions on Neural Networks between 1990 and 1991.

A biography and photo of **Panos Antsaklis** accompany the first article in this special issue.

Xiaojun Yang received the B.S. degree in mechanical and electrical engineering from Northwestern Textile Institute, Xian, China, in 1984, and the M.S. degree in electrical engineering from Hunan University, Hunan, China, in 1986. From 1986 to 1989, she worked as an engineer at Xian Machinery Manufacture Company in Electric Power. She obtained the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, in July 1992. She is currently a postdoctoral associate in the University of Notre Dame's department of electrical engineering. Her research interests are in the areas of complex systems modeling and control, intelligent control systems, and supervisory control of discrete event systems.

Costantino Lucisano was born in Africo Nuovo, Italy, on Aug. 5, 1965. He received his Laurea in Information Science from the Universita' degli Studi di Milano, Milan, Italy, in 1992. His thesis dissertation was in the area of neural networks for controlling non-linear distributed systems. During 1992, he was a research assistant at the Italian Cancer Research Center. Between 1993 nd 1994 he was a visiting scholar and is currently in the doctoral program at the University of Notre Dame's department of electrical engineering. His current research interests include intelligent control systems, learning, hybrid systems, automata, neural networks, and fuzzy logic.

Sampled Data

How to Kill a Professional Institute: Advice to Members

- Don't participate beyond paying your dues. Let "them" handle things. Then complain that members have no voice in management.
- Decline all offices and committee appointments—you're too busy. Then offer vociferous advice on how "they" should do things.
- If appointed to a committee, don't work—it's a courtesy appointment. Then complain because the Institute has stagnated.
- If you do attend committee meetings, don't initiate new ideas. Then you can play "devil's advocate" to those submitted by others.
- Don't rush to pay your dues—they're too high anyway. Then complain about poor financial management.
- Don't encourage others to become members. That's selling. Then complain that membership is not growing.
- Don't read the mail from headquarters—it's not important. Then complain that you're not kept informed.
- Don't volunteer yours talents—that's ego fulfillment. Then complain that you're never asked, never appreciated.
- And if by chance the Institute grows in spite of your contributions, grasp every opportunity to tell the new generation of members how tough it was; how hard you worked in the old days to bring the Institute to its present level of success.

— From the Editor's email