M.D. Lemmon, Xiaojun Yang and P. J. Antsaklis, "Inductive Inference of Logical DES Controllers using the L* Algorithm," Technical Report of the ISIS (Interdisciplinary Studies of Intelligent Systems) Group, No. ISIS-94-010, Univ of Notre Dame, September 1994.

Inductive Inference of Logical DES Controllers using the L^* Algorithm

Xiaojun Yang, Michael Lemmon, and Panos Antsaklis *
Department of Electrical Engineering
University of Notre Dame, Notre Dame, IN 46556

September 15, 1994

Abstract

Discrete event system (DES) controller synthesis requires that the plant deterministic finite automaton (DFA) be known. There are many situations when prior knowledge of the plant DFA is not realistic. The impracticality of knowing the plant DFA is particularly evident if the DFA is derived from examples of the plant's legal behaviours. It is well known that the problem of inferring a minimal DFA from positive examples is NP-complete, thereby casting doubt on the utility of traditional DES controller synthesis methods. Recently, however, an algorithm known as the L^* algorithm has been proposed for DFA inference. This algorithm has been shown to have polynomial computational complexity. This paper demonstrates how a modification of the L^* algorithm can be used for the on-line synthesis of DES controllers.

1 Introduction

Discrete event system (DES) controller synthesis methods as proposed by Ramadge and Wonham [1] require that the state transition model of the desired legal behaviours be known. LaFortune has pointed out [2] that this requirement is often unreasonable. First, the number of states in the DFA may grow exponentially in the number of constituent processes [3]. This situtation is observed in manufacturing systems when the machines and workpieces are modelled separately. Second, when it is impossible to determine when and where processes are terminated and initiated[4], the DES plant may be time-varying. The railway scheduling system is a typical instance of this situation. Third, the controlled DES model may not be known completely as is the case in DES plants derived from hybrid control systems [5]. Fourth, when legal behaviours are expressed as quasi-formal language specifications then construction of the associated DFA may be difficult if not impossible [6].

*The partial financial support of the National Science Foundation (MSS-9216559) is gratefully acknowledged

This paper uses an on-line inductive learning algorithm to synthesize the DES controller. Inductive inference [7] is a machine learning protocol which determines the minimal Boolean function consistent with a set of input-output pairs of that function. These input-output pairs are called "examples", so that inductive inference is sometimes referred to as learning by example. Inductive inference procedures provide a method for identifying DES controllers from given examples of the desired legal behaviours of the plant. In the context of the Ramadge-Wonham (RW) synthesis procedure, inductive inference algorithms identify the minimal deterministic finite automaton (DFA) consistent with examples of the plant's desired legal behaviours. Note that since the DFA is learned by "example", no explicit representation of the plant's legal behaviours is needed. Inductive learning methods simply require a reliable method for identifying whether or not a given string is legal.

The class of inductive learning methods being used are applicable to the inference of regular sets. There are two types of learning methods, passive and active. *Passive* learning methods "passively" observe examples of the regular set and base their determination on those observations. *Active* learning methods "actively" select certain strings and test their membership in the target set. Angluin [13] and Gold [15] have shown that the inference of minimal DFA's by passive observation of examples is NP-complete. See [14] for a recent survey of results on the inference of regular sets. These results were applied by Tsitsiklis [3] to show that the RW-synthesis is often impractical when the legal behaviours become extremely complex.

Recently, however, a learning procedure combining passive and active observation has been shown to have a computational complexity which is polynomial. The L^*-algorithm is a learning procedure proposed by Angluin [8] which uses passive examples and actively generated counterexamples to infer the minimal DFA for a regular set. This algorithm has been shown to be polynomial in the size of the minimal DFA and the length of the required counterexamples. This paper proposes using the L^*-procedure for the on-line synthesis of DES controllers. Because of the L^*-algorithm's polynomial complexity, it is expected that this procedure will provide a practical method for DES controller synthesis which answers some of the issues raised by Tsitsiklis [3] concerning the complexity of the RW-synthesis method.

The synthesis method introduced in this paper is an on-line synthesis procedure. On-line control of DES has been discussed previously in [9] [10] [11] [17] [16]. In the context of manufacturing systems, on-line schemes have been developed to solve deadlock avoidance problem[12]. This idea was developed into an on-line control scheme based on "limited lookahead control policies" [2]. That scheme determines the control action at each exectution based on an N-step ahead prediction of the controlled system's behaviour. Schapire [17] and R. Rivest [16] have used the L^* learning algorithm to infer finite automatons in robotic grid worlds. This work is related to ours in its use of the L^* algorithm. It does not, however, address controllability issues in the generation of the DES controller.

The objective of this paper is to show how the L^* algorithm can be used for the on-line synthesis of DES controllers. Because of the polynomial complexity of this algorithm, it may provide a computationally feasible approach to DES controller synthesis. The results in this paper are preliminary results providing examples of the proposed synthesis procedure. The remainder of this paper is

organized as follows. Section 2 summarizes the L^* learning algorithm as originally introduced by Angluin [8]. Section 3 discusses the modified L^* algorithm as we've used it. Section 4 presents examples drawn from problems first discussed in [1]. The examples provide concrete examples of the L^* learning algorithm's application in finding the supremal controllable sublanguage.

2 L^* Learning Algorithm

Let Σ denote the finite event alphabet, Σ^* denote all the finite event strings defined over Σ, and K denote the unknown regular set defined on Σ, $K \subseteq \Sigma^*$. Assume that we are given a set of *examples*, $\mathcal{E} = \{s_i\}$ $(i = 1, 2, \ldots)$ which are known to be element of K. It is well known [13] [15] [3] that to construct the minimal acceptor (DFA) consistent with these examples is NP-complete. Essentially this means that the problem of inductively inferring a regular language on the basis of passive examples is computationally intractable. This type of inference is sometimes referred to as "passive" learning since the learning system passively observes examples and takes no action to generate additional examples.

In [8], it was shown that a combination of active and passive learning will have a computational complexity that is polynomial in the size of the alphabet, the number of states in the minimal acceptor, and the length of the "counterexamples" which the learning system actively generates to probe the system. This type of learning process is "active" because the algorithm actively constructs counterexamples to the acceptor derived from the passively observed examples. The learning procedure introduced by Angluin was called the L^* learning procedure. Recent variations of this algorithm have suggested modifications improving the original method's convergence rate [17].

L^* learning is a procedure which methodically builds up an *observation table* from examples and counterexamples. The observation table is built up from strings $s \in \Sigma^*$. It is assumed that the algorithm is capable of determining whether or not a given string s is in the set K. The observation table consists of 3 parts, a non-empty finite prefix-closed set S of strings over Σ, a nonempty finite suffix-closed set E of strings over Σ, and a function $T : (S \cup S\Sigma)E \rightarrow \{0, 1\}$. The function T takes strings in $s \in (S \cup S\Sigma)$ onto 0 if they are not in K, otherwise the function T returns 1. This function is often called a *membership oracle*. The observation table is therefore characterized by the 3-tuple (S, E, T). The ith observation table constructed by the L^* algorithm will be denoted as T_i. It is a 2-dimensional array whose rows are labeled by strings $s \in S \cup S\Sigma$ and whose columns are labeled by symbols $\sigma \in E$. The entries in the labeled rows and columns are given by $T(s\sigma)$.

Let row(s) denote the table entries in the row labeled by string s. An observation table is said to be *closed* if for all $t \in S\Sigma$, there exists an $s \in S$ such that row(t) = row(s). An observation table is said to be *consistent* if there exist strings s_1 and s_2 in S such that row(s_1) = row(s_2), and for all $\sigma \in \Sigma$, row$(s_1\sigma)$ = row$(s_2\sigma)$. An observation table is said to be *complete* if it is closed and consistent. It has been shown [8] that a DFA $M(S, E, T) = \{Q, q_0, F, \delta\}$ can be constructed from the observation table by the following procedure.

$$Q = \{\text{row}(s) : s \in S\} \tag{1}$$

$$q_0 = \text{row}(\epsilon) \tag{2}$$

$$F = \{\text{row}(s) : (s \in S) \wedge (T(s) = 1)\} \tag{3}$$

$$\delta(\text{row}(s), \sigma) = \text{row}(s\sigma) \tag{4}$$

It is further known that this constructed DFA is the smallest acceptor for the regular set consistent with the positive examples of K used in the table.

The L^* learning procedure constructs the observation table for the minimal acceptor of K in the following manner. See [8] for more details.

L^* Learning Algorithm [8]

1. **Form initial observation table, T_0**
   Let $S = \varepsilon$ and $E = \varepsilon$.
   Use the *membership oracle* to evaluate $T_i = (S, E, T)$ where $i = 0$.

2. **Repeat:** While $T_i = (S, E, T)$ is not complete:

   (a) If T_i is not consistent,
   then find $s_1, s_2 \in S$, $\sigma \in \Sigma$, $e \in E$ such that $row(s_1) = row(s_2)$ and $T(s_1\sigma e) \neq T(s_2\sigma e)$,
   add σe to E,
   extend T to $(S \cup S\Sigma)E$ using queries to the *membership oracle*.

   (b) If T_i is not closed,
   then find $s_1 \in S$, $\sigma \in \Sigma$ such that
   $row(s_1\sigma)$ is different from all $row(s)$, $s \in S$,
   add $s_1\sigma$ to S,
   extend T to $(S \cup S\Sigma)E$ using queries to the *membership oracle*.

3. Once T_i is complete, let $M_i = M(T_i)$.
   Make the conjecture that M_i is the minimal acceptor of K.
   Ask the *counterexample oracle* about the conjecture's validity.
   The oracle returns a counterexample, $t \in \Sigma^*$ if the oracle declares the conjecture to be false.

   (a) Add t and all its prefixes to S.

   (b) Extend T to $(S \cup S\Sigma)E$ using queries to the *membership oracle*.

4. Set $i = i + 1$ and return to **Repeat** until the conjecture is declared true.

5. Halt and output M_i.

The running time of the L^* algorithm has been shown to be polynomial. Let n denote the minimal size for K, m the maximal length of a counterexample, and $l = |\Sigma|$. Then L^* can make at most $n-1$ incorrect conjectures. The maximum length of strings in E is $n-1$. The total number of strings in E is less than n. The maximum length of strings in S is $m+n-1$. The total number of strings in S can not exceed $n+m(n-1)$. The maximum length of strings in $(S \cup S\Sigma)E$ is $m+2n-1$. The maximum cardinality of $(S \cup S\Sigma)E$ is therefore at most $(l+1)(n+m(n-1))n$. This last result bounds the size of the observation table as a polynomial in m and n (see [8] for details).

3 L^*-Synthesis of DES Controller

Assume that the plant language, $L(G)$ and control specification, K, are described by regular languages and hence can be realized by finite automatons G and $M(K)$, respectively. The event symbols Σ are assumed to be partitioned into controllable Σ_c and uncontrollable event sets Σ_u, both of which are known. The plant event traces are assumed to be observable. Since G and $M(K)$ are not assumed to be known, the L^* learning algorithm has been proposed as a method for inductively synthesizing the optimal DES controller. The following subsections discuss the components of the proposed on-line DES synthesis method. Subsection 3.1 discusses the use of N-step lookahead windows. Subsections 3.2 and 3.3 discuss the membership and counterexample oracles, respectively. Subsection 3.4 discusses the on-line algorithm in which Angluin's L^* procedure is used to identify the optimal DES controller.

3.1 Lookahead Window

Rather than assuming that the plant automaton G is known, this paper assumes that plant knowledge is confined to a limited lookahead window of behaviours. This type of plant knowledge was used in [2]. Let $L(G,N,s)$ denote all strings of length not longer than N generated by the plant, G, after an observed trace s. Let $L_u(G,N,s)$ denote the set of uncontrollable traces and let $L_c(G,N,s)$ denote controllable traces in $L(G,N,s)$. Those traces in $L(G,N,s) \cap K$ whose controllability cannot be decided will be tagged as *pending*. The set of pending traces in $L(G,N,s)$ will be denoted as $L_p(G,N,s)$.

The preceding remarks partitioned $L(G,N,s)$ into three mutually exclusive sets. The following propositions characterize the controllable traces, $L_c(G,N,s)$ and uncontrollable traces, $L_u(G,N,s)$ in the prediction window. Pending traces are given by $L_p(G,N,s) = L(G,N,s) \cap K - L_u(G,N,s) - L_c(G,N,s)$. These propositions are stated without proof.

Proposition 1 *If there exists $t \in L(G,N,s) - K$, and it can be writen as $t = t_1 \sigma t_2$, where $\sigma \in \Sigma_u$, $t_1 \in L(G,N,s) \cap K$, $t_2 \in \Sigma_u^*$, then $t_1 \in L_u(G,N,s)$.*

Proposition 2 *Given $t \in (L(G,N,s) - L(G,N-1,s)) \cap K$, $\forall t_1 \in \Sigma_u^*$ s.t. $|u| \leq N_u(L(G))$. where $tt_1 = D_u(tt_1)u$, if $tt_1 \in K$, then $t \in L_c(G,N,s)$.*

Given $t \in L(G,N-1,s) \cap K$, $\forall t_1 \in \Sigma_u^* - \varepsilon$ s.t. $tt_1 \in L(G,N,s)$, if $tt_1 \in L(G,N-1,s) \cap K \cup L_c(G,N,s) \cap (L(G,N,s) - L(G,N-1,s))$, then $t \in L_c(G,N,s)$.

3.2 Membership Oracle

The control specification K is a prefix-closed regular set. Rather than knowing its acceptor, $M(K)$, this paper assumes that there exists a Boolean function $T : \Sigma^* \rightarrow \{0,1\}$ which declares whether or not a given string is a legal system behaviour. This mapping is called the *membership oracle*. It will be discussed in greater detail below. No specific assumptions are made about the implementation of T since specified behaviours may only be described by a quasi-formal set of "rules".

The basic membership oracle is a Boolean function mapping strings $t \in \Sigma^*$ onto $\{0,1\}$. If a string is is illegal or uncontrollable, then its value is 0, otherwise its value is 1. A formal algorithmic description of the membership oracle's outputs is given below. For $t \in \Sigma^*$, let $T(t|N,s)$ denote membership oracle.

$$T(t|N,s) = \begin{cases} 0 & \text{if } t \notin K \text{ or } t \in L_u(G,N,s) \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

Note that the membership oracle is defined with respect to the current prediction window, $L(G,N,s)$. The dependence of the membership oracle on the prediction window represents a major difference between the traditional L^* algorithm and our use of that algorithm. The dependence of the oracle on the prediction window is a reflection of the fact that the oracle cannot always decide string membership in the event of uncontrollable events. The introduction of the lookahead window helps reduce this uncertainty, but it does not remove it. The membership oracle therefore changes in a dynamic manner as the system evolves. Essentially it improves as the system evolves over time, being able to correctly declare set memberhsip for a larger and larger set of strings. This fact means that in implementations of the L^* procedure, the entries of the observation table will always have to be re-evaluated every time a new prediction window is generated. This dynamic aspect of the L^* implementation presented below represents a fundmental departure from traditional implementations of the algorithm.

3.3 Counterexample Oracle

Once the L^* algorithm has constructed a complete observation table, the procedure generates counterexamples to modify that table. The *counterexample oracle* is responsible for finding counterexamples to a given acceptor. Prior work [8] in L^* learning has used a sampling oracle to generate counterexamples. The counterexample oracle used in this paper is essentially a search procedure for efficiently exploring the acceptor's and the plant's behaviours.

The counterexample oracle is an algorithm which searches the acceptor and plant for illegal or uncontrollable and controllable behaviours. There are two ways in which a counterexample can be generated. First, searching acceptor generates counterexamples. When a string is illegal, but it can be generated by the acceptor, it is a counterexample. Second, searching prediction window generates counterexamples. If a uncontrollable string can be generated by the controller or a controllable string never be accepted by the controller, it should be used as counterexamples. The following procedure provides a systematic method for obtaining these strings. Counterexamples will be generated in

three different instances. These instances and their associated protocols are itemized below.

1. **CE Protocol 1**
   Generating counterexamples from an examination of the acceptor $M(T)$.
   Let $N_1 = (L(G) + 1)(|K| + 1)$ and let $i = 1$. Any $t \in L(M, i, s) - L(M, i-1, s)$ such that $t \notin K$ for $i = 1, \ldots, N_1$ is a counterexample.

2. **CE Protocol 2**
   Generating counterexamples from an examination of uncontrollable strings in the current prediction window.
   Any string $t \in L_u(G, N, s)$ will be a counterexample if $t \in M(T)$

3. **CE Protocol 3**
   Generating counterexamples from an examination of controllable strings in the current prediction window.
   Any string $t \in L_c(G, N, s)$ will be a counterexample if $t \notin M(T)$.

3.4 On-Line Synthesis Procedure

This section summarizes our implementation of the L^* algorithm when using the time-varying memberhsip oracle discussed in subsection 3.2. The following procedure also explicitly shows how we use the three cases itemized in subsection 3.3 to search for counterexamples of a conjectured acceptor.

Since we obtain examples on-line and improve membership oracle on-line too, L^* algorithm can not be directly used in on-line controller synthesis. However, we do make use of the basic idea of L^* algorithm, by which complete observation table is constructed efficiently and counterexamples are used to derive a correct conjecture so that the corresponding minimal constructure of the regular set is learned. In the following, the on-line controller synthesis algorithm is given.

1. Given initial prediction window $L(G, N, \varepsilon)$, search for $L_u(G, N, \varepsilon)$, then build initial observation table according to membership oracle $T(t|N, \varepsilon)$. Let $s = \varepsilon$
2. When the table is not complete, make the table complete as L^* algorithm does according to current membership oracle.
3. When the acceptor is derived from the complete table, search CE-protocol 1 in this acceptor for the shortest one. Add it into S to get a new complete table in the same way as 2. Repeat searching with CE-protocol 1 to obtain the new acceptor. Continue until no more counterexamples are found.
4. Use CE-protocol 2 to search the acceptor for the shortest counterexample. If t is the returned counterexample, add $D_u(t)$ into S to get a new complete table in the same way as 2. Repeat until no more counterexamples are found.

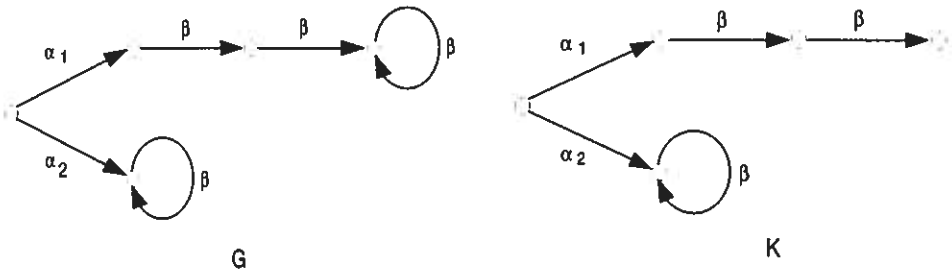


Figure 1: Automatons of $L(G)$ and K (example 1)

5. Compute $L_c(G, N, s)$ and use CE-protocol 3 to search the acceptor for the shortest counterexample, t. Add t into S to get a new complete table in the same way as 2. Repeat until no more counterexamples are found.

6. Using the acceptor as a controller, denote the enabled event set as $C = L(M, 1, s)$. If $C \cap L(G, 1, s) = \emptyset$, or $|s| \geq (|L(G)|+1)(|K|+1)$, reset the system to the initial state, then generate a new path according to $C = L(M, 1, \varepsilon)$. Otherwise, randomly choose one event $\sigma \in C$, assume the system execute σ. Let $s \Longleftarrow s\sigma$, getting new prediction window.

7. Computing new uncontrollable string in $L(G, N, s)$, update membership oracle accordingly. That is, if there exists $t \in SE \cap L_u(G, N, \bar{s})$, s.t. $T(t) = 1$, then it is changed into $T(t) = 0$. Go to 2 to repeat the algorithm.

8. When no more counterexamples are found or the system has generated enough events, then the algorithm ends.

3.5 Simple Example

This is a simple example used to illustrate how the on-line controller synthesis algorithm works. In this example, the DES plant and specification are shown in figure 1. The event set is $\Sigma = \{\alpha_1, \alpha_2, \beta\}$ and the uncontrollable event set, $\Sigma_u = \{\beta\}$. For this example, $L(G) = \overline{(\alpha_1\beta^2 + \alpha_2)\beta^*}$ and the control specification is $K = \overline{\alpha_1\beta^2 + \alpha_2\beta^*}$. Initially, of course, these automatons are not known to the controller.

The specification is assumed to be given as a set of quasi-formal rules. In this example, the specification rules are;

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.

In addition to these rules, it is assumed that the plant's future behaviour can be predicted for up to $N = 2$ steps in the future.

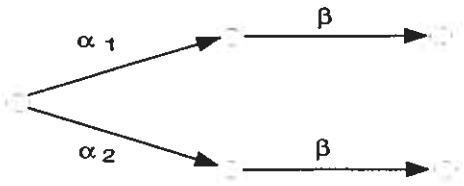


Figure 2: $L(G, 2, \varepsilon)$ for example 1

To illustrate the generation of the plant's predicted behaviour, assume that the system starts in its initial state. The first observed transition is therefore the null trace, ε. The language $L(G, 2, \varepsilon)$ is represented by the tree shown in figure 2. Note that in this tree, all of the predicted behaviours are legal (with respect to our aforementioned rules), but that they all terminate in an uncontrollable event, β. The controllability of these events in the window therefore cannot be decided. Consequently all predicted traces in the window are pending except ε ($L_p(G, 2, \varepsilon) = L(G, 2. \varepsilon) - \varepsilon$).

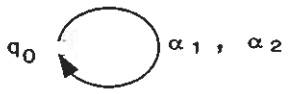
Using the fact that all predicted traces are legal, the initial observation table T_0 and its acceptor $M(T_0)$ are readily written down as shown in the first row of figure 3. Figure 3 shows all of the complete observation tables T_i and their associated acceptors $M_i = M(T_i)$ generated by the L^* algorithm for this particular example.

Note that while trace $\alpha_1\alpha_2$ can be generated by M_0, this trace clearly fails to satisfy K. Therefore $\alpha_1\alpha_2$ is a counterexample. The counterexample is used to modify T_0 by first adding α_1 to S. The resulting table, however, is not consistent because $\text{row}(\varepsilon) = \text{row}(\alpha_1)$ and $T(\varepsilon\alpha_1) = 1$, $T(\alpha_1\alpha_1) = 0$. The event α_1 therefore has to be added into E. The resulting table, T_1 is now complete. This table and its acceptor are shown in figure 3.

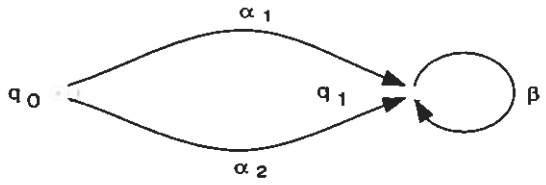
The process is repeated by attempting to generate another counterexample. Note that $\alpha_1\beta\beta\beta \notin K$ but it is accepted by $M(T_1)$. This trace is therefore a counterexample. The traces $\alpha_1\beta$ and $\alpha_1\beta\beta$ are added into S. To force consistency, β and $\beta\beta$ are added to E. The complete observation table, T_2 and its acceptor are shown in figure 3.

Note that since $M_2 \subset K$, no counterexamples can be generated from the acceptor. To search for additional counterexamples, traces from the plant need to be examined. This is done by using the current controller $M(T_2)$ and observing the plant's resulting behaviour. Assume, for example, that the application of M_2 generates the trace α_2. Let $G|M_2$ denote the plant DES G controlled by acceptor M_2. The controlled lookahead language, $L(G|M_2, N, \alpha_2)$, consists of a single trace, $\alpha_2\beta\beta$. This trace is accepted by M_2 and hence is legal (i.e. not a counterexample). Letting the system evolve one more step, the observed trace becomes $\alpha_2\beta$. The two step ahead prediction once again consists of a single trace, $\alpha_2\beta^3$. Once again, this string accepted by M and hence is not a counterexample. Note that by continuing in this manner, all further traces generated by the controlled plant would be legal. The preceding discussion highlights a specific point about using these learning methods for on-line DES synthesis. It is quite possible that these languages are "non-ergodic". In other words, for a given initial trace, it is possible that certain legal traces will never be generated by the plant. These other legal traces would only have been generated if the plant had started with a different

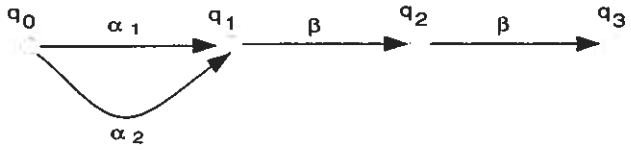
T_0	ε
ε	1
α_1	1
α_2	1



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



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



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

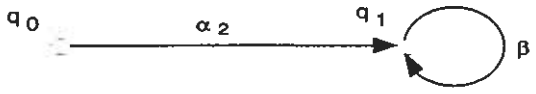


Figure 3: Completed Observation Tables and their Acceptors (example 1)

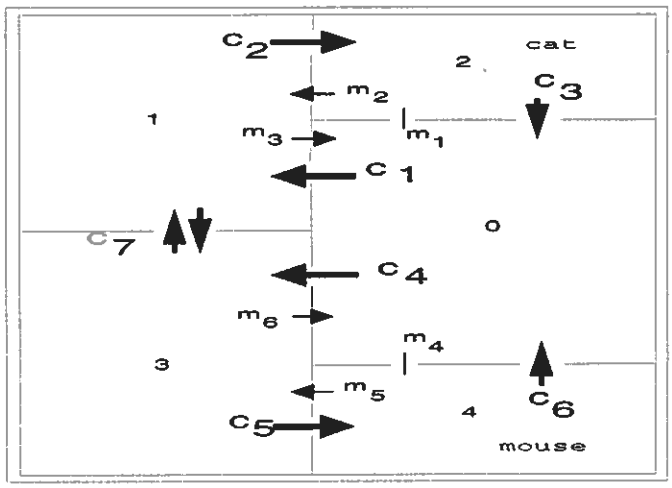


Figure 4: Cat and Mouse Problem

initial event string. This fact can be easily seen in 1, where the first event is either α_1 or α_2. The initial testing of M_2 was done assuming an initial event α_2. Now assume that $\alpha_1\beta$ was the first event generated by the plant. In this case, the two step-ahead trace will be $\alpha_1\beta^3$. This behaviour is illegal and can therefore be used as a counterexample to update T_2. The completed observation table, T_3, and its acceptor are shown in figure 3.

Note that this last acceptor M_3 is the supremal controllable sublanguage for this problem as computed in [1]. Any further attempts, therefore, to generate counterexamples would be futile.

4 Examples

This section demonstrates L^* learning on two different problems drawn from [1]. The first problem is the well-known cat and mouse problem. The second problem is based on a manufacturing application.

4.1 Cat and Mouse Problem

This classic example provides another complex problem which the L^* learning algorithm is able to solve. The event set of the system is $\Sigma = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, m_1, m_2, m_3, m_4, m_5, m_6\}$, where $\Sigma_u = \{c_7\}$. The events mark the entry/exit of the cat (c) or mouse (m) from a room. The underlying structure of the rooms and the labeled transitions are shown in figure 4. The control specification is to find that control scheme which permits the cat and the mouse the greatest possible freedom of movement while guaranteeing that (1) the cat and mouse never occupy the same room simultaneously and that (2) both are able to return to the initial state. In this example, we assume a prediction window, N, of size 2. Let (i, j) denote that the cat and mouse are in rooms i and j, respectively. The initial state is $(2, 4)$.

To construct the initial observation table, first construct the tree representing $L(G, 2, \varepsilon)$ and

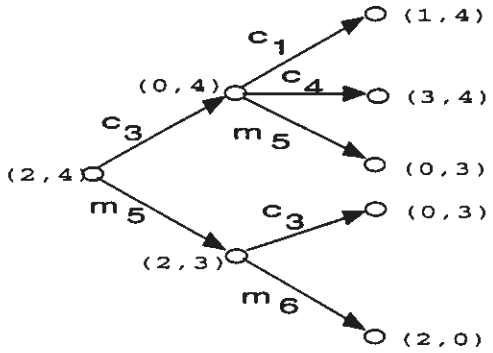


Figure 5: Prediction Window $L(G, 2, \varepsilon)$ (example 2)

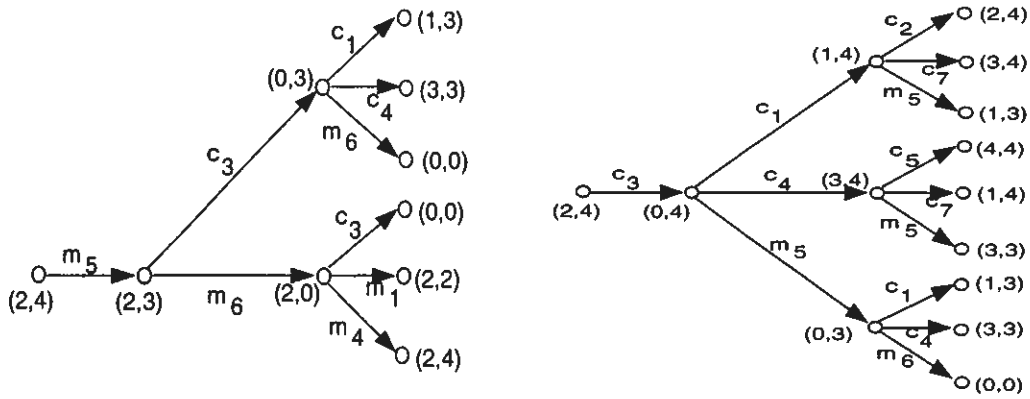


Figure 6: Prediction Windows $L(G|M_0, 2, m_5)$ and $L(G|M_0, 2, c_3)$ (example 2)

examine its traces. This tree is shown in figure 5. In this window, ε, c_3, and m_5 are controllable strings. The other strings are all pending. The entries in the initial table are decided according to our quasi-formal specification. The resulting observation table, T_0 and its acceptor are shown in figure 7.

Note that this acceptor will generate no illegal behaviours and accepts all observed controllable strings. M_0 is therefore used as a controller in an attempt to generate system behaviours. Suppose the first generated event is that the mouse moves, m_5. The prediction window $L(G|M_0, 2, m_5)$ is shown in figure 6. In this window, $m_5 m_6$ is controllable but not accepted by M_0. This string is used as a counter-example to modify T_0, and thereby obtain the observation table T_1. This table and its acceptor are shown in figure 7. Note that this control strategy works by allowing the cat to move to one room and then trapping him there.

Now use M_1 as a controller and see what happens if the cat moves first, i.e. the event c_3 is issured. In this case the prediction window is shown in figure 6. At first glance, there are no counterexamples in this window. Note, however, that it is possible to have an uninterrupted string of c_7 transitions. Note that if this occurs then the cat and mouse always occupy separate rooms.

The strings $c_3c_1c_7c_7^*$ and $c_3c_4c_7c_7^*$ should therefore be treated as controllable. In other words, c_3c_1 and c_3c_4 are counterexamples for T_0. This is the first set of sufficient conditions for identifying uncontrollable traces in proposition 2 of subsection 3.1. The counterexample c_3c_1 modifies the observation table T_1 to obtain T_2. This observation table and its acceptor are shown in figure 7

Note that c_3c_4 is accepted by M_2. Using M_2 as a controller shows that if traces $m_5c_3c_4$ or $m_5c_3c_1c_7$ occur, then the system arrives at the illegal state (3,3). Furthermore c_7 is uncontrollable so that $m_5c_3c_1$ is an uncontrollable string. Therefore $m_5c_3c_4$, $m_5c_3c_1$ should not be generated. In addition, if the prefix m_5c_3 of these previous strings is executed, then the cat and the mouse never return to the initial state. So m_5c_3, c_3m_5 should be used as counterexamples to generate the observation table T_3 and its associated acceptor (see figure 7). Proceeding as before using M_3 as a controller, yields counterexample of controllable string $c_3c_4c_7$. Modifying T_3 with this trace produces a complete observation table T_4. This table and its acceptor are shown in figure 7. Note that the final controller is the supremal controllable sublanguage for this problem.

4.2 Manufacturing System Problem

In this example two machines are connected by a buffer. The event set and uncontrollable event sets are $\Sigma = \{\alpha_1, \alpha_2, \beta_1, \beta_2\}$, $\Sigma_u = \{\beta_1, \beta_2\}$. Event α_i means that machine i starts working, β_i means machine i finishes working. The automaton for $L(G)$ is shown in figure 8. In this example, the system states have the following interpretation. I_i means machine i is idle and W_i means machine i is working. The buffer B has one slot with two states, E, empty and F, full. Initially the system is in state (I_1, I_2).

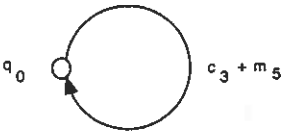
The control specification is to keep the buffer at o or 1 at all times. Let $|\alpha|(t)$ denote the number of occurances of α in t, where $\alpha \in \Sigma, t \in \Sigma^*$. The control specification can be expressed as $K = \{u : u \in \Sigma^*, |\alpha_2|(t) \leq |\beta_1|(t) \leq |\alpha_2|(t) + 1, \forall t \in \bar{u}\}$. This means that α_2 and β_1 strictly alternate and that β_1 occurs first. Note that K is not included in the system behaviour $L(G)$. At this point, it is different from the previous examples. Its model is shown in figure 8.

Since $N_u(L(G)) = 2$, assume that the size of the prediction window is 3. In the initial step, the current trace is $s = \varepsilon$. The initial prediction window is shown in figure 9. Note that in this window the events ε, α_1, $\alpha_1\beta_1$ are controllable, $\alpha_2\Sigma^*$ and $\alpha_1\alpha_2\Sigma^*$ are illegal. All other traces are pending. There are no uncontrollable strings.

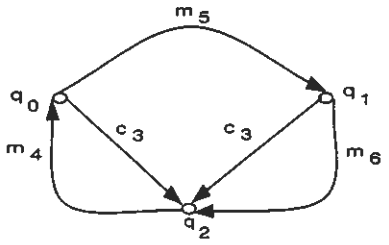
In the same way as above, we generate the initial observation table and its acceptor, as shown in figure 11. Trace $\beta_1\beta_1$ is illegal, but it can be generated by the initial acceptor, so it is a counterexample. β_1 is added into the table T_0. To make the table consistent, α_2 is added into E. The resulting observation table T_1 is complete. This table and its acceptor are shown in figure 11.

This acceptor M_1 is the model of control specification K, so no more counterexample is found by searching this acceptor. Since the available controllable strings in the initial prediction window can be generated by this acceptor, and no uncontrollable string is found, the acceptor can be used as a controller. Using M_1 to generate the enabled event set $L(M_1, 1, \varepsilon) \cup \Sigma_u$, assume that the observed event trace is α_1. The 3-step ahead prediction window is shown in the figure 10. Note that $\alpha_1\beta_1\alpha_1\beta_1$

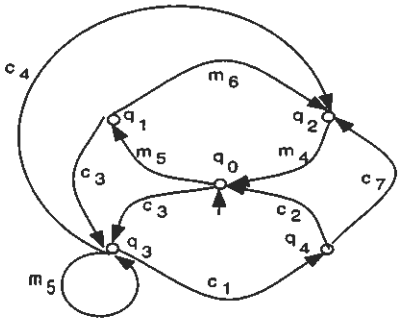
T_0	ε
ε	1
c_3	1
m_5	1



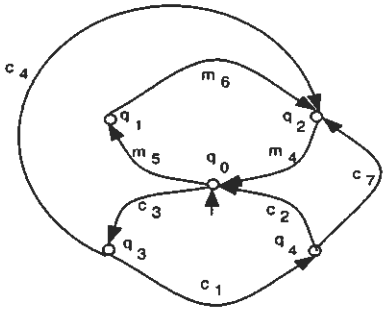
T_1	ε	m_6	c_3
ε	1	0	1
m_5	1	1	1
m_5m_6	1	0	0
c_3	1	0	0
m_5c_3	1	0	0
$m_5m_6m_4$	1	0	1



T_2	ε	m_6	c_3	c_1	c_2
ε	1	0	1	0	0
m_5	1	1	1	0	0
m_5m_6	1	0	0	0	0
c_3	1	0	0	1	0
c_3c_1	1	0	0	0	1
m_5c_3	1	0	0	1	0
$m_5m_6m_4$	1	0	1	0	0
c_3c_4	1	0	0	0	0
c_3m_5	1	0	0	1	0
$c_3c_1c_2$	1	0	1	0	0
$c_3c_1c_7$	1	0	0	0	0



T_3	ε	m_6	c_3	c_1	c_2
ε	1	0	1	0	0
m_5	1	1	1	0	0
m_5m_6	1	0	0	0	0
c_3	1	0	0	1	0
c_3c_1	1	0	0	0	1
$m_5m_6m_1$	1	0	1	0	0
c_3c_4	1	0	0	0	0
$c_3c_1c_2$	1	0	1	0	0
$c_3c_1c_7$	1	0	0	0	0



T_4	ε	m_6	c_3	c_1	c_2	m_4
ε	1	0	1	0	0	0
m_5	1	1	1	0	0	0
m_5m_6	1	0	0	0	0	1
c_3	1	0	0	1	0	0
c_3c_1	1	0	0	0	1	0
c_3c_4	1	0	0	0	0	0
$c_3c_4c_7$	1	0	0	0	1	0
$m_5m_6m_4$	1	0	1	0	0	0
$c_3c_1c_2$	1	0	1	0	0	0
$c_3c_1c_7$	1	0	0	0	0	0
$c_3c_4c_7c_2$	1	0	1	0	0	0
$c_3c_4c_7c_7$	1	0	0	0	0	0
$c_3c_4c_7m_5$	1	1	0	0	1	0

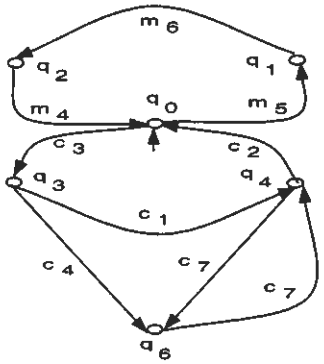


Figure 7: Completed Observation Tables and Acceptors (Example 2)

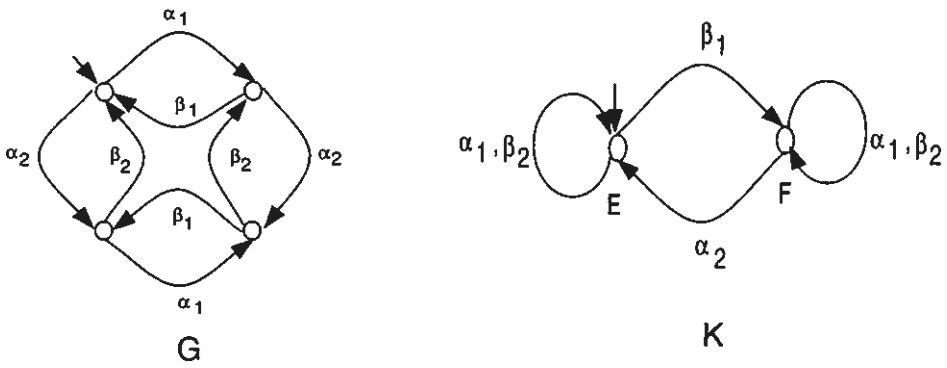


Figure 8: Automatons for L(G) and K (example 3)

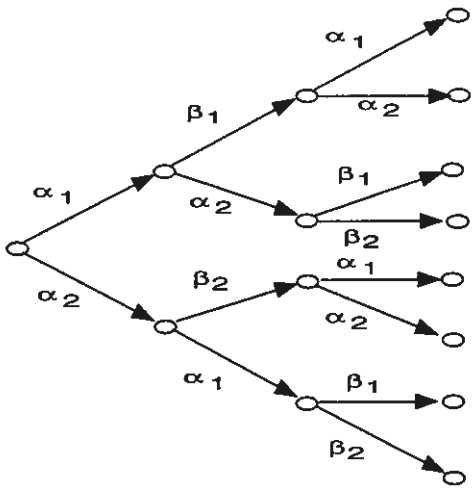


Figure 9: First Prediction Window ($L(G,3,\varepsilon)$) (example 3)

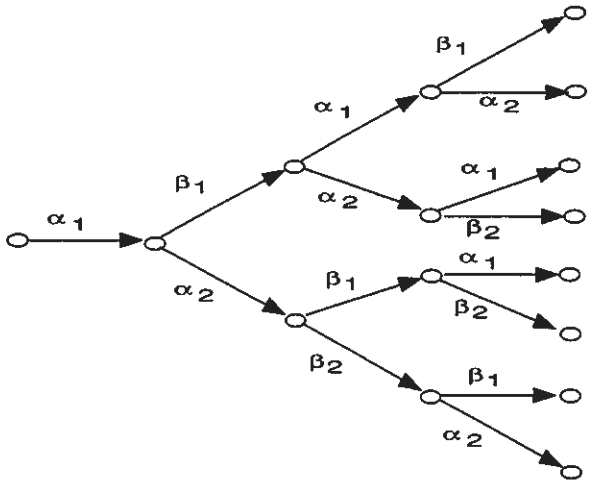


Figure 10: $L(G|M_1, 3, \alpha_1)$ (example 3)

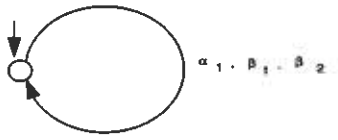
is illegal and β_1 is uncontrollable. Therefore $\alpha_1\beta_1\alpha_1$ is an uncontrollable trace which we don't want accepted by our controller. But this trace is accepted by M_1, so it must be a counterexample. α_1, $\alpha_1\beta_1$ are added into S. α_1, $\beta_1\alpha_1$ are added into E to make the table consistent. The resulting table, T_2, is still complete and is shown in figure 11.

The strings $\alpha_1\beta_1\alpha_2\alpha_1$, $\alpha_1\beta_1\alpha_2\beta_2\alpha_1$, $\alpha_1\beta_1\alpha_2\alpha_1\beta_1\beta_2\alpha_2\alpha_1\beta_2$ can be generated by M_2. From the prediction windows for them, the following string $\alpha_1\beta_1\alpha_2\alpha_1\beta_1\alpha_1$, $\alpha_1\beta_1\alpha_2\beta_2\alpha_1\beta_1\alpha_1$ and $\alpha_1\beta_1\alpha_2\alpha_1\beta_1\beta_2\alpha_2\alpha_1\beta_2\beta_1\alpha_1$ are identified as uncontrollable strings. In addition to this observation, we can see from the prediction windows of $\alpha_1\beta_1\alpha_2\alpha_1\beta_1$ and $\alpha_1\beta_1\alpha_2\alpha_1\beta_2$, that the strings $\alpha_1\beta_1\alpha_2\alpha_1\beta_1\beta_2\alpha_1$ and $\alpha_1\beta_1\alpha_2\alpha_1\beta_2\beta_1\alpha_1$ are uncontrollable. These traces should therefore be used as counterexamples since they can be generated by M_2, Therefore the strings $\alpha_1\beta_1\alpha_2$, $\alpha_1\beta_1\alpha_2\alpha_1$, $\alpha_1\beta_1\alpha_2\alpha_1\beta_1$, $\alpha_1\beta_1\alpha_2\alpha_1\beta_1\beta_2$ are added into S. The strings $\beta_2\beta_1\alpha_1$, $\alpha_1\beta_2\beta_1\alpha_1$, $\beta_2\alpha_1$ are put into E to make the table consistent. The table entries are now re-evaluated using the updated membership oracle The resulting complete observation table T_3 and its acceptor M_3 are shown in figure 11.

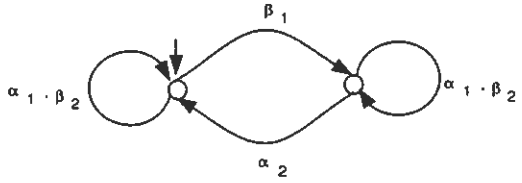
Since $L(M_3)\cap L(G)$ is equal to the supremal controllable language obtained in [1], so this acceptor is the optimal controller being learned. Although its structure is different from that in [1], the behaviour of closed-loop logic control system is identical. The reason for the difference is that the set of legal non-system behaviours is not empty. Our procedure, however, allows these strings to be accepted by the controller and so they will be learned. The consequence is that the learned DES controller will not necessarily be the minimal optimal controller.

The preceding remarks suggest that our original procedure does not always generate the minimal optimal controller. We therefore experimented with different strategies for controling the size of the learned controller. The proposed strategy involves assigning values to table entries in a slightly different way. In particular, we would assign table entries to distinguish between legal system and non-system behaviours. In particular, if t is illegal or uncontrollable its value is 0. If t is a legal

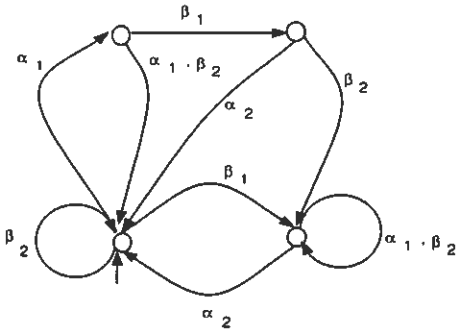
T_0	ε
ε	1
α_1	1
β_1	1
β_2	1



T_1	ε	α_2
ε	1	0
β_1	1	1
α_1	1	0
β_2	1	0
$\beta_1\alpha_1$	1	1
$\beta_1\alpha_2$	1	0
$\beta_1\beta_2$	1	1



T_2	ε	α_2	α_1	$\beta_1\alpha_1$
ε	1	0	1	1
β_1	1	1	1	0
α_1	1	0	1	0
$\alpha_1\beta_1$	1	1	0	0
β_2	1	0	1	1
$\beta_1\alpha_1$	1	1	1	0
$\beta_1\alpha_2$	1	0	1	1
$\beta_1\beta_2$	1	1	1	0
$\alpha_1\alpha_1$	1	0	1	1
$\alpha_1\beta_2$	1	0	1	1
$\alpha_1\beta_1\alpha_2$	1	0	1	1
$\alpha_1\beta_1\beta_2$	1	1	1	0



T_3	ε	α_2	α_1	$\beta_1\alpha_1$	$\beta_2\beta_1\alpha_1$	$\alpha_1\beta_2\beta_1\alpha_1$	$\beta_2\alpha_1$
ε	1	0	1	1	1	1	1
β_1	1	1	1	0	0	0	1
α_1	1	0	1	0	1	1	1
$\alpha_1\beta_1$	1	1	0	0	0	0	1
$\alpha_1\beta_1\alpha_2$	1	0	1	1	1	0	1
$\alpha_1\beta_1\alpha_2\alpha_1$	1	0	1	0	0	1	1
$\alpha_1\beta_1\alpha_2\alpha_1\beta_1$	1	1	0	0	0	0	0
$\alpha_1\beta_1\alpha_2\alpha_1\beta_1\beta_2$	1	1	0	0	0	0	1
β_2	1	0	1	1	1	1	1
$\beta_1\alpha_1$	1	1	1	0	0	0	1
$\beta_1\alpha_2$	1	0	1	1	1	1	1
$\beta_1\beta_2$	1	1	1	0	0	0	1
$\alpha_1\alpha_1$	1	0	1	1	1	1	1
$\alpha_1\beta_2$	1	0	1	1	1	1	1
$\alpha_1\beta_1\beta_2$	1	1	1	0	0	0	1
$\alpha_1\beta_1\alpha_2\beta_1$	1	1	1	0	0	0	1
$\alpha_1\beta_1\alpha_2\beta_2$	1	0	1	1	1	1	1
$\alpha_1\beta_1\alpha_2\alpha_1\alpha_1$	1	0	1	1	1	1	1
$\alpha_1\beta_1\alpha_2\alpha_1\beta_2$	1	0	1	0	1	1	1
$\alpha_1\beta_1\alpha_2\alpha_1\beta_1\alpha_2$	1	0	1	1	1	1	1
$\alpha_1\beta_1\alpha_2\alpha_1\beta_1\beta_2\alpha_2$	1	0	1	1	1	0	1
$\alpha_1\beta_1\alpha_2\alpha_1\beta_1\beta_2\beta_2$	1	1	1	0	0	0	1

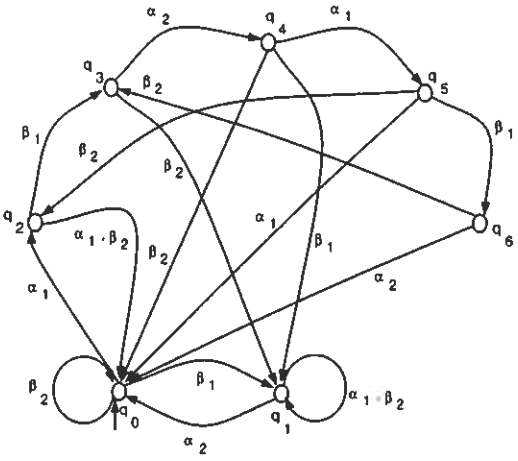


Figure 11: Completed observation tables and acceptors (example 3)

system behaviour, its value is 1. When t is legal, *but a non-system behaviour*, its value can be 1 or 0. This is because its acceptance by the controller does not influence the closed-loop system's logical behaviour. Such strings are assigned the value $*1$ to mark that they are legal non-system behaviours. Later, if changing $*1$ to 0 can make the table complete, then the $*1$ entries are set to 0. This generates a smaller observation table because it doesn't add additional strings to S and E as our original procedure would have done. When a string is legal, but it cannot be determined whether or not it is a system behaviour, we assign it the value $\dagger 1$. When future system behaviours are examined, some of these $\dagger 1$ entries can be changed into 1 or $*1$ if the strings are legal system or non-system behaviour, respectively.

Figure 12 shows the sequence of observation tables and acceptors that were generated by using the proposed strategy. As can be seen, the first two tables, T_0 and T_1 are the same. However, table T_2 shown in figure 12 is much simpler than the corresponding table in figure 11. This is because our marking strategy was able to avoid adding two additional columns to the table, thereby reducing table size. Note that the final acceptor generated by our strategy is indeed the optimal controller for this problem and corresponds to the minimal controller found in [1].

5 Discussion

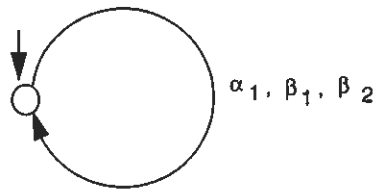
This paper has presented a method for on-line synthesis of DES controllers based on Angluin's L^* algorithm. Three specific examples were used to demonstrate that the proposed algorithm can compute the optimal controller. The motivation in using this method is three fold. First of all, the method does not require full knowledge of the plant generator. Instead this algorithm only assumed that finite lookahead behaviour of the system is known. In addition to this, the algorithm does not require a formal description of the control specification. The algorithm appears to work when the specification is stated as a quasi-formal specification. This is appropriate for cases in which control specifications may have arisen from human operator guidelines. Finally, the approach is well known to have a polynomial computational complexity, thereby answering initial reservations voiced [3] about the NP-complete nature of DES controller synthesis.

The results in the paper, however, are preliminary and empirical. This paper has demonstrated that in certain cases the L^* algorithm can synthesize the optimal controller in an on-line manner. How general this approach is, however, has yet to be rigorously proven.

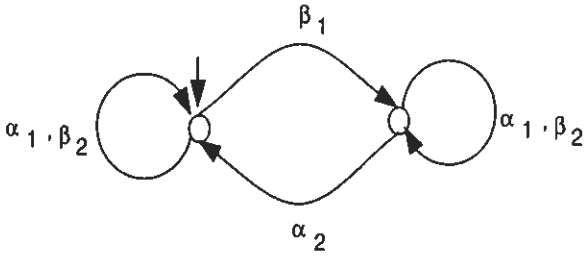
A particular feature of the approach is its handling of uncontrollable events and its use of a dynamic membership oracle. This clearly distinguishes our work from that in traditional L^* applications where the membership oracle is fixed. How the use of dynamic oracles affect convergence rate and computational complexity is an issue for future study.

The examples provide considerable insight into how well-known machine learning protocols can be modified for use in DES synthesis. How inclusive or general these strategies are, needs to be examined in greater detail. We have a good understanding of the utility and generality of certain strategies proposed in this paper. However, the strategy proposed for controlling controller complexity is strictly ad hoc in nature and its utility still needs to be examined.

T_0	ε
ε	1
α_1	1
β_1	*1
β_2	*1



T_1	ε	α_2
ε	1	0
β_1	*1	*1
α_1	1	0
β_2	*1	0
$\beta_1\alpha_1$	*1	*1
$\beta_1\alpha_2$	*1	0
$\beta_1\beta_2$	*1	*1



T_2	ε	α_2
ε	1	0
β_1	*1	*1
α_1	1	0
$\alpha_1\beta_1$	1	1
β_2	*1	0
$\beta_1\alpha_2$	*1	0
$\beta_1\beta_2$	*1	*1
$\alpha_1\alpha_1$	*1	0
$\alpha_1\beta_2$	*1	0
$\alpha_1\beta_1\alpha_2$	1	0
$\alpha_1\beta_1\beta_2$	*1	*1

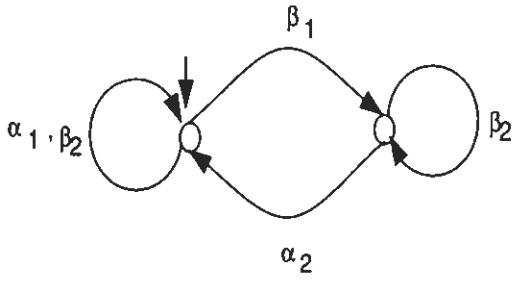


Figure 12: Completed observation tables and acceptors (example 3)

References

[1] 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.

[2] Sheng-Luen Chung, Stéphane Lafortune, "Limited lookahead policies in supervisory control of discrete event systems", *IEEE Trans. on Automatic Control*, Vol. 37, No. 12, pp. 1921–1935, Dec. 1992.

[3] 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.

[4] Z. Banaszak and B.H. Krogh, "Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows", *IEEE Trans. Robot. Automat.*, Vol. 6, No. 6, pp. 724–734, Dec. 1990.

[5] M.D. Lemmon, J.A. Stiver, and P.J. Antsaklis, "Learning to coordinate control policies of hybrid systems", *Proceedings of the American Control Conference*, pp. 31–35, San Francisco, CA, June 1993.

[6] C.A. Brooks, R. Cieslak and P. Varaiya, "A method for specifying, implementing, and verifying media access control protocols", *IEEE Contr. Syst. Mag.*, Vol. 10, No. 4, pp. 87–94, June 1990.

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

[8] D. Angluin, "Learning regular sets from queries and counterexamples", *Int. J. Information and Computation*, Vol. 75, No. 1, pp. 87–106, 1987.

[9] D.P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Englewood Cliffs, NJ: Prentice-Hall, 1987.

[10] K.M. Passino and P.J. Antsaklis, "A system and control theoretic perspective on artificial intelligence planning systems", *Int. J. Appl. Artificial Intelligence*, Vol. 3, No. 1, pp. 1–32, 1989.

[11] B.H. Krogh and D. Feng, "Dynamic generation of subgoals for autonomous mobile robots using local feedback information", *IEEE Trans. Automat. Contr.*, Vol. 34, No. 5, pp. 483–493, May 1989.

[12] Y. Viswanadham, Y. Narahari and T.L. Johnson, "Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models", *IEEE Trans. Robot. Automat.*, Vol. 6, No. 6, pp. 713–723, Dec. 1990.

[13] D. Angluin, "On the complexity of minimum inference of regular sets", *Int. J. Information and Control*, Vol. 39, pp.337–350, 1978.

[14] L. Pitt, "Inductive inference, DFAs, and computational complexity", *Lecture Notes in Artifical Intelligence*, Vol. 397, J. Siekmann, 1990.

[15] E. Mark Gold, "Complexity of automaton identification from given data", *Int. J. Information and Control*, Vol. 37, pp.302–320, 1978.

[16] R. Rivest and R. Schapire, " Diversity-based inference of finite automata", In *28th Annual Symposium on FOundations of COmputer Science*, pp. 78-87, October 1987

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