Proceedings of the
American Control Conference
Seattle, Washington • June 1995

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

Xiaojun Yang, Michael Lemmon [1], and Panos Antsaklis
Department of Electrical Engineering
University of Notre Dame, Notre Dame, IN 46556
lemmon@maddog.ee.nd.edu

Abstract

It is well known that learning procedures such as the L^* algorithm will infer minimal deterministic finite automatons (DFA) in polynomial time through the use of membership and equivalence queries. This paper introduces a modification of the L^*-algorithm that can be used for the inductively inferring optimal logical DES controllers in which prior knowledge of the plant is confined to a finite lookahead window of predicted behaviours.

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. This requirement is not satisfied when the DES plant is only partially known. This paper proposes dealing with plant uncertainty by using inductive learning algorithms. In particular,this paper uses a modification of the L^*-algorithm [2] to infer the supremal controllable sublanguage of the unknown plant. There is already a great deal of prior work attempting to account for imperfectly known DES plants. The prior work most relevant to this paper's technique is found in [3] and [4].

Inductive inference [5] is a machine learning protocol which determines the minimal boolean functional 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. A query concerning whether or not a given input-output pair is a positive or negative example of the functional will be called a *membership query.* A query concerning whether an hypothesized language is equivalent to another language will be called an *equivalence query.* Algorithms answering the membership or equivalence queries will be called *oracles.* The inductive learning procedure will use a combination of equivalence and membership queries to construct a boolean functional that approximates the unknown boolean concept in an appropriately defined sense [6].

The class of learning methods being used in this paper are applicable to the inference of regular sets. There are a large number of positive and negative results concerning the inference of minimal DFA's. It has been shown that inference of minimal DFA's through membership oracles is NP-complete, [7]. The learning problem, however, exhibits polyonomial complexity if membership and equivalence queries are used. The L^*-algorithm [2] has been shown to have a computational complexity that is polynomial in the size of the minimal DFA and the number of equivalence queries.

The objective of this paper is to show how membership and equivalence queries can be used for on-line synthesis of DES controllers. In particular, we are interested in seeing if it is possible to inductively learn the supremal controllable sublanguage. Direct application of the L^*-algorithm, however, is not possible due to uncontrollable events. Uncontrollable events introduce uncertainty into the algorithm's membership queries. Since the original L^* algorithm assumes complete membership oracles, this means that a modified version of the L^* algorithm will need to be developed. This paper introduces such a modification of the L^* procedure. Preliminary proofs of the algorithm's convergence will be found in [8]. This paper presents the algorithm and provides an example illustrating its useage.

[1]The partial financial financial support of the National Science Foundation (MSS92-16559) and the Electric Power Research Institute (RP8030-06) are gratefully acknowledged.

2. L^* Algorithm

Let Σ denote the finite event alphabet, Σ^* denote all the finite event strings defined over Σ, and K denote an unknown regular set defined on Σ, $K \subseteq \Sigma^*$. Assume there exists a set of *examples*, $\mathcal{E} = \{s_i\}$, $(i = 1, 2, \ldots)$ which are known to be elements of K.

Inductive inference of regular sets is based on the simple observation that if two strings s_1 and s_2 arrive at the same state, then for any suffix, e, the strings s_1e and s_2e will also arrive at the same state. This observation is an *aggregation principle* that aggregates all strings in the example set arriving at the same state. It can also be used to differentiate between different states in the DFA. Using this principle, it makes sense to decompose all strings of the example set into a prefix-closed set of strings, S, and suffix closed set of strings , E. Consider a boolean functional, $T : \Sigma^* \rightarrow \{0, 1\}$ which maps a string $s \in \Sigma^*$ onto 0 if it is in K and 1 otherwise. T is called a *membership oracle* and can be represented by an *observation table*. Each row in the table is labeled with a string in $S \cup S\Sigma$ and each column in the table is labeled with a string in E. The table entry associated with the row labeled s and the column labeled e will be the value of the functional, $T(se)$.

The preceding aggregation principle can be used to inductively build up an observation table. Let row(s) denote an operator mapping the string, s, labeling a row onto the entries of that row. By the aggregation principle, if there exist two strings s_1 and s_2 in S such that when $\text{row}(s_1) = \text{row}(s_2)$ and there exists $e \in \Sigma$ such that $\text{row}(s_1e) \neq \text{row}(s_2e)$, then these two strings cannot have arrived at the same state. This fact is used to identify new states in the DFA. An observation table that satisfies the aggregation principle will be said to be *consistent*. In particular, the observation table is said to be *consistent* if there exist strings s_1 and s_2 in S such that $\text{row}(s_1) = \text{row}(s_2)$ and for all $\sigma \in \Sigma$, $\text{row}(s_1\sigma) = \text{row}(s_2\sigma)$. The resulting table is said to be *closed* if for all $t \in S\Sigma$, there exists an $s \in S$ such that $\text{row}(s) = \text{row}(t)$. An observation table is *complete* if it is closed and consistent. It has been shown [2] that a minimal DFA can be constructed from a complete observation table. The minimal DFA, $M(S, E, T) = \{Q, q_0, \delta\}$ is characterized by $Q = \{\text{row}(s) : s \in S\}$, $q_0 = \text{row}(\epsilon)$, and $\delta(\text{row}(s), \sigma) = \text{row}(s\sigma)$.

Unfortnately, it is well known [7] that building up a complete observation table solely on the basis of membership queries will be NP-complete. The L^* algorithm therefore uses equivalence queries and membership queries to yield a polynomial time algorithm [2]. An *equivalence query* hypothesizes that the constructed acceptor generates the desired language, K. The oracle then returns true (1), if the acceptor generates K and false (0) otherwise. In the event that the equivalence query gets a negative response, then the oracle also returns a *counter-example* to the hypothesis. The new information contained in the counter-example is then added to the observation table through membership queries.

The L^* procedure constructs an observation table for the minimal acceptor in the following manner.

Step 1: Form initial observation table.
Let $i = 0$, $S = \epsilon$, and $E = \epsilon$. Complete the table entries for table T_i by calls to the membership oracle.

Step 2: Complete the observation table, T_i.
If T_i is not consistent, then add entries to E to make it consistent
If T_i is not closed, then add entries to S to close it.

Step 3: Equivalence Query
Construct the minimal DFA, M_i, for the completed observation table
Make an equivalence query about the validity of M_i. If the oracle returns a counter-example, $t \in \Sigma^*$, then add t and all its prefixes to S. Extend the table to $(S \cup S\Sigma)E$ using membership queries.

Step 4: Loop
Let $i = i + 1$ and go back to step 2 until the equivalence query returns no other counter-examples.

3. Uncertain Membership Oracle

The L^* algorithm requires unambiguous answers to the membership query. This assumption is overly restrictive since there may be insufficient information to answer all membership queries. An example of this situation is found in the inference of DES controllers with uncontrollable events. Assume that the plant language, $L(G)$, and the control specification, K, are described by regular languages and 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, Σ_u, event sets, both of which are known. The objective is to identify the supremal controllable sublanguage [1] by observing the controlled plant's behaviour. To use the L^* procedure, however, we must be able to identify whether a given string, s, generated by the plant belongs to K. In the event that s terminates in a controllable event, then the membership of s can be decided unambiguously. If, however, s terminates in an uncontrollable event, then the membership of s

can only be determined by examining the future behaviour of the plant. This may be impossible to do and so the membership oracle returns one of three responses, TRUE (1), FALSE (0), and AMBIGUOUS (*). Therefore in order to apply the L^* procedure to on-line controller synthesis, we must modify the procedure to deal with incomplete or uncertain membership oracles.

This section introduces a modification of the L^* algorithm which can be used for DES controller synthesis. The resulting algorithm uses an uncertain membership oracle based on a limited lookahead window [3]. 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 $L(G, N, s)$. As noted above, the controllability of some strings in $L(G, N, s) \cap K$ may not be decidable. These undecidable strings will be referred to as *pending strings*. It is therefore reasonable to propose the following membership oracle,

$$T(t|N,s) = \begin{cases} 0 & \text{if } t \in L_u(G,N,s)\Sigma^* \\ 1 & \text{if } t \in L_c(G,N,s) \\ * & \text{otherwise} \end{cases} \quad (1)$$

The basic operations in the L^* algorithm involve splitting the state associated with a specific row in the observation table. If a table is inconsistent, then there exist strings in E which can be used to split a row into several distinct rows, i.e. different states. The chief complication that uncertain membership oracles introduce is that it is no longer clear whether or not a given row should be split. In this situation, therefore, it makes sense to only split rows when the available data supports the splitting. In other words, when there exist uncertain entries in the observation table, row splitting and aggregation should only be based on the deterministic entries in the observation table. This simple principle provides a systematic method for constructing an observation table which is consistent with the available data and yet retains sufficient flexibility to refine itself when new data (i.e. counter-examples) become available. The following definition introduces this notion of row aggregation.

Definition 1 *Let* **S** *denote the set of row labels,* $\{s_i\}$. *Consider the mth subset,* S_m, *of* S *such that for any* s_i *and* s_j *in* S_m *and for all* $e \in E$, *the following conditions hold;* $T(s_i e) \neq *$, $T(s_j e) \neq *$, *and* $T(s_i e) = T(s_j e)$. *The collection* S_m *is called the mth* aggregated group *of the observation table.*

If there is no other row label in S that can be added to the aggregated group S_m, then the aggregated group is said to be maximal. According to the aggregation principle, this implies that the aggregated group corresponds to a state of the finite automaton. It is now possible to define concepts of closure and consistency for aggregated groups in a similar way to what is found in the L^* procedure.

Definition 2 *Let* S_i $(i = 1, \ldots, M)$ *denote the ith maximally aggregated group for a given observation table. If for all* $s \in S\Sigma - S$, *there exists* $s_1 \in S$ *and* i *such that* $s_1 \in S_i$ *and* $s \in S_i$, *then the observation table is said to be* closed.

Consider all pairs of strings s_1 *and* s_2 *such that for some* i, $s_1 \in S_i$ *and* $s_2 \in S_i$. *If for all* $\sigma \in \Sigma$ *there exists* j *such that* $s_1\sigma \in S_j$ *and* $s_2\sigma \in S_j$, *or* $s_1\sigma \notin S_j$ *and* $s_2\sigma \notin S_j$ *for all* j, *then the observation table is said to be* consistent.

The preceding definitions determine a closed and consistent observation table with regard to aggregated groups of rows, rather than individual rows. It is a straightforward generalization of the definitions of consistency and closure used in the L^* procedure. Essentially, this approach postpones the splitting of aggregated states unitl there is sufficient information available to make a reliable splitting. Procedures for completing the observation table are done in the same way that the L^* algorithm completes its observation table. The acceptor, $M(S, E, T) = \{Q, q_0, \delta\}$, is then derived from the maximally aggregated groups in the table through the following formulas, $Q = \{S_i, i = 0, \ldots, m\}$, $q_0 = S_0$, and $\delta(S_i, \sigma) = \{S_{j1}, S_{j2}, \ldots, S_{jk}\}$ where for all $s \in S_i \cap S$, $s\sigma \in \bigcap_{l=1}^{k} S_{jl}$.

Note that in the event that there is no ambiguity in the membership query, then the modified L^* procedure is identical to the original L^* algorithm. Further note that there may be cases when a single row belongs to several different maximal aggregated groups. In this case, the resulting acceptor may exhibit nondeterministic transitions.

Preliminary proofs for the convergence of the modified L^* procedure will be found in [8]. The proof involves carefully looking at three different types aggregated groups and showing how the counter-example returned by an equivalence query will always bring new information into the observation table allowing splitting of aggregated groups. If the size of the minimal DFA is known to be bounded, then this implies that after a finite number of counter-examples the modified L^* procedure will have formed the minimal number of aggregated states for the unknown DFA. The controller that is then learned appears to be a superlanguage $\bar{K}$, of the specification language.

This superlanguage contains the supremal controllable sublanguage as well as a collection of non-system behaviours. The intersection of $\bar{K}$ with the original plant behaviours can be shown to be the supremal controllable sublanguage.

4. DES Controller Synthesis: example

This section demonstrates the modified L^* learning procedure on a manufacturing application drawn from [1]. 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 that machine i finishes working. 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 0 or 1 at all times. Let $|\alpha|(t)$ denote the number of occurrences of α in string t ($\alpha \in \Sigma, t \in \Sigma^*$). 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.

We assume a lookahead window of length 3. In the initial step, the current trace is $s = \varepsilon$. Inside prediction window, $L(G, 3, \varepsilon)$, the events ε, α_1, and $\alpha_1\beta_1$ are controllable, whereas $\alpha_2\Sigma^* \cap L(G, 3, \varepsilon)$ and $\alpha_1\alpha_2\Sigma^* \cap L(G, 3, \varepsilon)$ are uncontrollable. All other traces in the window are pending. The initial observation table, T_0, is shown below,

T_0	ε
ε	1
α_1	1
β_1	*
β_2	*

The acceptor, M_0, is shown in figure 1. Since the controllable strings in the initial lookahead window can be generated by M_0 and all of the uncontrollable strings are not accepted by M_0, there is no counter-example in the lookahead window and we use M_0 as the controller. Assume that α_1 is the observed event trace under controller M_0. The 3 step ahead prediction window $L(G, 3, \alpha_1)$ has illegal string $\alpha_1\beta_1\alpha_1\beta_1$. Since β_1 is uncontrollable, this means that trace $\alpha_1\beta_1\alpha_1$ is an uncontrollable trace which shouldn't be accepted by the controller. This trace, however, is accepted by M_0 and so it is used as a counter-example. The traces α_1 and $\alpha_1\beta_1$ are therefore added to S and α_1 is added to E. The resulting completed table is

T_1	ε	α_1
ε	1	1
α_1	1	*
$\alpha_1\beta_1$	1	0
$\beta_1 \cup \beta_2$	*	*
$\alpha_1(\alpha_1 \cup \beta_2)$	*	*
$\alpha_1\beta_1(\beta_1 \cup \alpha_2 \cup \beta_2)$	*	*

Since there are no more counter-examples in the lookahead window, M_1 can be used as the new controller. In this case, the observed trace becomes $\alpha_1\beta_1$. In the new lookahead window, $L(G, 3, \alpha_1\beta_1)$, there are two strings which are controllable but not accepted by M_1. These strings are $\alpha_1\beta_1\alpha_2\beta_2$ and $\alpha_1\beta_1\alpha_2$ and they are therefore treated as counter-examples. The observation table, T_2, resulting from adding these two counter-examples is,

T_2	ε	α_1	α_2
ε	1	1	0
α_1	1	*	0
$\alpha_1\beta_1$	1	0	1
$\alpha_1\beta_1\alpha_2$	1	*	*
$\beta_1 \cup \beta_2$	*	*	*
$\alpha_1(\alpha_1 \cup \beta_2)$	*	*	*
$\alpha_1\beta_1(\beta_1 \cup \beta_2)$	*	*	*
$\alpha_1\beta_1\alpha_2\beta_2$	1	*	*
$\alpha_1\beta_1\alpha_2(\Sigma - \beta_2)$	*	*	*

Figure 1 shows the associated acceptor, M_2. Assume that this acceptor generates the illegal system behaviour $\alpha_1\beta_1\alpha_2\beta_2\alpha_2$. Since $\alpha_1\beta_1\alpha_2\beta_2$ is controllable, this string is added into table T_2 to obtain table T_3,

T_3	ε	α_1	α_2
ε	1	1	0
α_1	1	*	0
$\alpha_1\beta_1$	1	0	1
$\alpha_1\beta_1\alpha_2$	1	*	*
$\alpha_1\beta_1\alpha_2\beta_2$	1	*	0
$\beta_1 \cup \beta_2$	*	*	*
$\alpha_1(\alpha_1 \cup \beta_2)$	*	*	*
$\alpha_1\beta_1(\beta_1 \cup \beta_2)$	*	*	*
$\alpha_1\beta_1\alpha_2(\Sigma - \beta_2)$	*	*	*
$\alpha_1\beta_1\alpha_2\beta_2(\Sigma - \alpha_2)$	*	*	*

Use M_3 as a controller and assume that the observed trace is $\alpha_1\beta_1\alpha_2\alpha_1$. Analysis of the lookahead window leads to the observation table T_4,

T_4	ε	α_1	α_2
ε	1	1	0
α_1	1	*	0
$\alpha_1\beta_1$	1	0	1
$\alpha_1\beta_1\alpha_2$	1	1	*
$\alpha_1\beta_1\alpha_2\beta_2$	1	*	0
$\beta_1 \cup \beta_2$	*	*	*
$\alpha_1(\alpha_1 \cup \beta_2)$	*	*	*
$\alpha_1\beta_1(\beta_1 \cup \beta_2)$	*	*	*
$\alpha_1\beta_1\alpha_2(\Sigma - \beta_2)$	*	*	*
$\alpha_1\beta_1\alpha_2\beta_2(\Sigma - \alpha_2)$	*	*	*

Note that $L(M_4) \cap L(G)$ equals the supremal controllable sublanguage obtained in [1], so this acceptor is the optimal controller to be learned. The structure of this controller is different from that in [1]. In spite of this difference however, the behaviour of both closed loop logical systems is identical. The differences arise because the controller learned by our modified L^* procedure also accepts a variety of non-system behaviours which cannot be generated by the plant. The additional non-system behaviours in M_4 arise from the fact that the counter-examples are all chosen from system behaviours (legal and illegal). Without a source of counter-examples representing non-system behaviours, the learning algorithm will be unable to remove these extra behaviours from M_4.

5. Summary

This paper presented a method for the on-line synthesis of DES controllers based on Angluin's L^* algorithm. The method does not require full knowledge of the plant generator. Our knowledge is only restricted to a finite window of lookahead behaviours. In addition to this, the algorithm does not require a formal description of the control specifications. The algorithm can use quasi-formal specifications. This is appropriate for cases in which supervisory specifications arise from human operator guidelines.

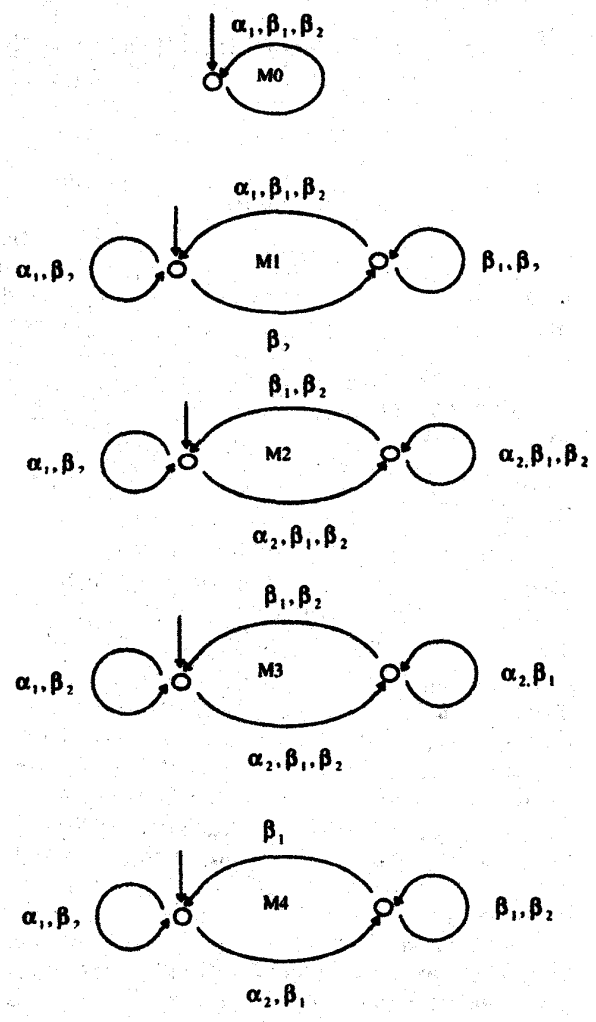

Figure 1: Completed Acceptors

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

[3] 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.

[4] S. Young, Vijay Garg, "transition uncertainty in discrete event systems", *Proceedings 6th IEEE International Symposium on Intelligent Control*, pp. 245-250, Arlington, VA. Aug. 1991.

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

[6] L. Valiant, "A Theory of the Learnable", *Comm. of the ACM*, Vol 27:1134-1142, 1984.

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

[8] X. Yang, M. Lemmon, and P. Antsaklis, "inductive inference of logical DES controllers using the L^* algorithm", Technical Report ISIS-95-00x, University of Notre Dame, Notre Dame, IN, May 1995.