4-1983

A Learning Strategy for a Class of Probabilistic Automata

Lyle Amos Reibling

*Western Michigan University*

Follow this and additional works at: [http://scholarworks.wmich.edu/masters_theses](http://scholarworks.wmich.edu/masters_theses)

Part of the *Computer Sciences Commons*

**Recommended Citation**


[http://scholarworks.wmich.edu/masters_theses/1618](http://scholarworks.wmich.edu/masters_theses/1618)

This Masters Thesis—Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
A LEARNING STRATEGY FOR A CLASS OF PROBABILISTIC AUTOMATA

by

Lyle Amos Reibling

A Thesis Submitted to the Faculty of The Graduate College in partial fulfillment of the requirements for the Degree of Master of Science Department of Computer Science

Western Michigan University Kalamazoo, Michigan April 1983
A LEARNING STRATEGY FOR A CLASS OF PROBABILISTIC AUTOMATA

Lyle Amos Reibling, M.S.

Western Michigan University, 1983

A learning strategy is developed for a class of probabilistic automata. Given an input sequence to the probabilistic automaton, and the output sequence arising from the probabilistic transition function of the automaton, the learning strategy will conjecture the probabilistic automata which cover the input-output sequence observed and will infer the probabilistic transition structure of the conjectured automata.
ACKNOWLEDGEMENTS

Several people deserve recognition on behalf of the author for whom their advice and counsel was indispensible: Dr. Michael Olinger for his counsel on the proper scope of the problem, and Richard Corey for many stimulating discussions on the nature of the research. Also deserving of mention is Don Gray for his graciousness in making accommodations in the work schedule for the completion of the research, and David Dainelis for his aid in the use of the computer resources of the Simulation Facilities of Lear Siegler Instrument Division.

Above all, to the author's wife, Anne, and children, a special word of gratitude for their longsuffering faith and endurance in the completion of this thesis.

Lyle Amos Reibling
INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
REIBLING, LYLE AMOS

A LEARNING STRATEGY FOR A CLASS OF PROBABILISTIC AUTOMATA

WESTERN MICHIGAN UNIVERSITY M.S. 1983

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy.
Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages
2. Colored illustrations, paper or print
3. Photographs with dark background
4. Illustrations are poor copy
5. Pages with black marks, not original copy
6. Print shows through as there is text on both sides of page
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements
9. Tightly bound copy with print lost in spine
10. Computer printout pages with indistinct print
11. Page(s) lacking when material received, and not available from school or author.
12. Page(s) seem to be missing in numbering only as text follows.
13. Two pages numbered. Text follows.
14. Curling and wrinkled pages
15. Other

University Microfilms International

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS................................................................. ii

LIST OF FIGURES................................................................. iv

LIST OF TABLES................................................................. v

Chapter

I. INTRODUCTION................................................................. 1

II. REVIEW OF SELECTED LITERATURE........................................ 4

III. BACKGROUND: FUNDAMENTALS OF AUTOMATA THEORY............... 19

IV. PROBLEM SCOPE, METHODOLOGY, AND NOTATION..................... 31

V. DETERMINISTIC LEARNING STRATEGY DESIGN
   AND IMPLEMENTATION...................................................... 47

VI. PROBABILISTIC LEARNING STRATEGY DESIGN
   AND IMPLEMENTATION..................................................... 62

VII. COMPLEXITY OF THE LEARNING STRATEGY............................. 79

VIII. CONCLUSIONS AND RECOMMENDATIONS................................. 97

APPENDIX A. IMPLEMENTATION LISTINGS FOR
             DETERMINISTIC CLASS........................................... 101

APPENDIX B. IMPLEMENTATION LISTINGS FOR
             PROBABILISTIC CLASS............................................ 120

BIBLIOGRAPHY................................................................. 151

iii
LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Moore's Simple Experiment</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Moore's Multiple Experiment</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Trace from M(2,2,2)</td>
<td>51</td>
</tr>
<tr>
<td>4</td>
<td>Partial Trace from P(2,2,2)</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>Problem Space Width Complexity</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>Level D1 Trace of P(∞,∞,2)</td>
<td>85</td>
</tr>
<tr>
<td>7</td>
<td>Level D4 Trace of P(∞,∞,2)</td>
<td>86</td>
</tr>
<tr>
<td>8</td>
<td>Problem Space Hierarchy of P(∞,∞,r)</td>
<td>92</td>
</tr>
<tr>
<td>9</td>
<td>LSPM Problem Space Complexity of P(p,q,r)</td>
<td>93</td>
</tr>
</tbody>
</table>
LIST OF TABLES

TABLE

1 Values of N  .................................................. 91

K
CHAPTER I

INTRODUCTION

Background

The learning of structure, or associations, between elements of a system is an interesting subject in Artificial Intelligence. Such an example is the inference of the internal structure of a black-box from examination of its output response to input stimuli. Another example is learning a computer program from the input-output traces of the program. Methods can be developed for conjecturing the internal structure of the black-box, and if one a-priori knows that the black-box is one of a class of black-boxes whose structure can be characterized, then additional criteria for the process of inferring the structure of the black box is available. In fact, the a-priori knowledge of the black box may dictate what type of conjectural machine model is necessary to characterize its behavior (Turing machine vs. pushdown automata vs. finite state machines). Various characterizations of the black-box can be envisioned. The equivalent automata types of the Chomsky hierarchy of languages can be one characterization.
Research Goals

In this research, a learning strategy is developed for a class of probabilistic automata. Given an input sequence to the probabilistic automaton, and the output sequence arising from the probabilistic transition function of the automaton, the learning strategy will conjecture the probabilistic automata which covers the input-output sequence observed and will infer the probabilistic transition structure of the conjectured automata. Thus the behavior of the black box is covered by the inference of machines which individually covers the behavior of the machine in the black box.

Chapter Review

Chapter 2 is a review of selected literature which has dealt with the concept of learning systems. Chapter 3 is a review of some fundamentals of automata theory to provide a background for the reader who is unfamiliar with finite state machines. In Chapter 4, the scope of the problem is reviewed and the assumptions for this research are specified. The methodology for this research is also discussed. Also, the notation used in this research is presented. An implementation for the learning strategy of the class M(p,q,r) of deterministic automata is presented in Chapter 5. This concept is
extended to a learning strategy of the class $P(p,q,r)$ of probabilistic automata. In Chapter 6 the algorithm LSPM is developed which implements the learning strategy for probabilistic automata. Chapter 7 discusses the problem space of LSPM and the complexity of the learning strategy.
CHAPTER II

REVIEW OF SELECTED LITERATURE

One of the classic papers in the field of automata theory is (Moore 1956). The motivation for (Moore 1956) was the determination of conclusions about the internal conditions of finite state automata from external experiments of the automata. The automata under study in this paper were deterministic finite state machines (DFSM). The DFSM considered have the output symbols associated with the states of the automata (the Moore model of a DFSM).

The external experiments consisted of supplying an input symbol sequence to the DFSM and observing the output symbol sequence from the DFSM in order to investigate questions about the internal structure of the DFSM. In the simple experiment, a copy of the DFSM receives a sequence of input symbols from an experimenter, which then makes some conclusion about the internal conditions of the DFSM after observing the sequence of output symbols emitted by the DFSM in response to the input symbol sequence. The conclusion of the experimenter is required to depend only on the input symbol sequence and the observed output symbol sequence.
of the DFSM.

**Figure 1: Moore's Simple Experiment**

In the multiple experiment, \( K \) copies of the DFSM are available to the experimenter. Each copy of the DFSM is in the same internal state prior to the experiment. The experimenter then sends an input symbol sequence to each of the \( K \) DFSM copies (the input symbol sequences should not be the same) and observes each DFSM's output symbol sequence.
Figure 2: Moore’s Multiple Experiment

The input symbol sequence to the DFSM does not need to be predetermined, but may depend on previous output of the DFSM, which allows branching in the experimenter.

Theorem 2 of (Moore 1956) states that it is never possible to perform experiments on a completely unknown machine which identifies that machine from the class of all sequential machines. However, with a restricted class from the class of all sequential machines it is possible to identify an unknown machine. Some of the
restrictions imposed on the class of all sequential machines are to specify the number of input and output symbols in the input and output symbol sets, respectively, of all machines in the restricted class of sequential machines, and to specify an upper bound on the number of distinguishable states of these machines. Other restrictions that are usually applied to the class of all sequential machines are to require that all the machines are completely specified (no state has undefined transitions for any input symbol of the input symbol set), and connected (all states of the machine can be reached by some input symbol sequence).

Other theorems established by (Moore 1956) that are concerned with the length of experiments are:

(1) distinguishing the states of a machine,
(2) distinguishing the states of different machines,
(3) identifying the state of a machine at the end of an experiment, and
(4) distinguishing a machine from all other machines in the same restricted class of all sequential machines.

Further problems for investigation which were suggested by (Moore 1956) are experiments with multiple input/output symbol sequences, where the multiple symbol sequence is a sequence of symbols which are grouped into
In this research, the multiple experiment concept of using $K$ copies of a FSM is implemented by serially applying the $K$ input symbol sequences to a single copy of the probabilistic finite state machine (PFSM). However, this necessitates the ability to reset the PFSM to its initial state between each of the $K$ input symbol sequences. This is equivalent to multiple experiments described previously which used multiple copies of the same machine. Also, in this research, the PFSM has outputs which are associated with the transitions (the Mealy model of finite state machines) rather than the state associated outputs.

The question posed by the work of (deLeeuw, et.al. 1956) is whether anything that can "be done" by a machine with a random element exists that cannot be done by a deterministic machine. The results of (deLeeuw, et.al. 1956) refer to possible enumeration of infinite sets and computation of infinite sequences, not to finite symbol sets and computation sequences. Therefore, the complexity of probabilistic machines performing a finite task was not addressed in (deLeeuw, et.al. 1956). A definition is offered in (deLeeuw, et.al. 1956) for probabilistic machine equivalence:
Two objects that are stochastic machines will be said to be equivalent if for any finite set of symbols

\[ s_1, \ldots, s_n \]

they have the same probability of eventually having that set included in their output.

In this research, rather than investigating the equivalence of a deterministic automaton operating in a random environment with a probabilistic automaton with an infinite symbol sequence, the learning of PFSM's which cover the input/output behavior of a finite sequence of symbols whose probability distribution is determined by the PFSM under test is investigated.

In (Gill 1961), the machine identification problems of (Moore 1956) were characterized into two problems:

1. Distinguishing: The problem is to find the current state from a known machine state, and
2. Homing: Given known machine states, the problem is to put the machine into a known state.

(Gill 1961) also formally extends the types of experiments into two types:

1. Preset: The input symbol sequence is completely designed in advance of the experiment and is valid for all initial states of the machine, and,
(2) Adaptive: Each subsequence of the input symbol sequence (with the exception of the first) is designed based on the previous subsequence of the output symbols of the machine.

Using these formal extensions, many gaps that were previously existing in the problem of state identification of finite automata were filled by theorems established by (Gill 1961).

In this research, all the experiments are preset, that is, not adaptive to the output symbol sequence of the PFSM.

In (Booth and Thompson 1973), the concept of probability is applied to formal language grammars. In their work, each production of a grammar, G, has a probability associated with the production. Probabilities are then defined for each word in the language, L(G), which is generated from the grammar, G. Properties of these probabilistic languages are then developed by (Booth and Thompson 1973).

Some definitions developed include a word function, which is a mapping from any word of a language to a real number; the word function is then defined to be measurable if the sum of the word functions over all
words of the language is some global maximum, \( N \), that is less than infinity; and this measurable word function is then said to be a probabilistic word function if \( N = 1 \). A word function is defined to be a computable word function if there exists an algorithm that can be used to compute the word function over its entire domain of words of the language.

In this research, the model under investigation is the probabilistic finite state machine. However, since a right-linear grammar is equivalent to a deterministic finite automata (and a construction procedure exists to convert between these equivalent models), the probabilities of the productions in the probabilistic grammar, \( G \), can be directly applied to the corresponding state transitions during the construction of a probabilistic finite automata from the right-linear grammar, \( G \), that has probabilistic productions.

(Thompson 1974) builds upon the previous work of (Booth and Thompson 1973) in researching probability-measure languages (pm-language) and probabilistic grammars (p-grammars). (Thompson 1974) defines a finite state probability string transducer as a non-deterministic finite state machine.
\[ S = ( Q, V, Z, d, q_0 ) \]

where \( Q \) is a finite nonempty set of states,

\( V \) is an input alphabet,

\( T \)

\( Z \) is an output alphabet,

\( d \) is a mapping: \( d(Q \times V) \rightarrow 2^T \)

\( q_0 \) in \( Q \), the start state

A pm-language is said to be fs (finite state) tractable if its word function, \( p(L) \), is specified by a finite state probability string transducer. (Thompson 1974) also defines equivalence as: two p-grammars \( G \) and \( G' \) are equivalent if

\[ L(G) = L(G') \text{ with } p(L(G)) = p(L(G')). \]

(Thompson 1974) also defines a transformation \( T \) on a p-grammar \( G \) as a procedure that produces a different p-grammar \( G' = T(G) \) such that \( G \) and \( G' \) are equivalent.

Four equivalence transformations that are introduced and explained by (Thompson 1974) are composition, decomposition, intermediation, and symbol normalization.
In (Thompson 1974), the concept of a finite state probability string transducer is similar to the probabilistic finite state machine (PFSM) that is the subject of investigation of this research. The definition of that string transducer differs from this PFSM in that the string transducer of (Thompson 1974) operates by transforming the input string which has a probabilistic distribution as the result of some p-grammar productions into an output string which also then has a probabilistic distribution, whereas the input to the PFSM has no origin in a probabilistic generation process.

The research by (Maryanski and Booth 1977) assumes that the source which is to be inferred is unknown and generates strings according to the rules of a p-grammar. A sample of the source consists of $M \gg 1$ strings that are statistically independent. A sample set is defined to be

$$S = \{ (x_1, f_1), (x_2, f_2), \ldots, (x_u, f_u) \}$$

where $x_i$ in $L(G)$ are the observed distinct strings, and $f_i$ are the number of times $x_i$ occurred.

The $f_i$'s are the frequency (of occurrence) counts.
(Maryanski and Booth 1977) has defined a sample set from \(L(G)\) to be structurally complete if each rule of the grammar \(G\) is used at least once in the generation of the sample strings. They also claim that if a grammar is to be inferred from a sample set of strings, the sample set must be structurally complete.

The inference process of (Maryanski and Booth 1977) is as follows: The probabilistic source to be inferred generates finite length strings upon request of the experimenter. The experimenter observes the sample set \(S\) which contains \(M\) strings, and determines the \(p\)-grammar modelling properties of the source based on information in the sample set \(S\). To solve the inference it is necessary to have a denumerable set of candidate grammars:

\[
G = \{ G_1, G_2, \ldots, G_j, \ldots \}
\]

The acceptable set to the probabilistic source may be obtained where \(G\) is finite and each candidate grammar in \(G\) is a deterministic finite state \(p\)-grammar. The grammars of \(G\) are tested in order until one of them is found in which the expected number of occurrences of each string does not markedly differ from the observed frequency of occurrence. The chi-square test is used as
a measure of variation to determine if a grammar in \( G \) satisfactorably models the sample set \( S \).

In (Maryanski and Booth 1977), maximum likelihood estimation techniques are used in the estimation of rule probabilities for an unambiguous finite state grammar. Both the rule set, \( R \), and rule probabilities, \( P \), of the grammar are partitioned into \( m \) partitions such that each partition \( R_i \) and \( P_i \) have associated with them all the rules with the same premise (the non-terminal antecedent of a p-grammar production rule), \( A_j \).

The number of times the rule \( r_i \) with the premise \( A_j \) is used is denoted \( n_{i,j} \), and \( N_{j} \) is the number of times the non-terminal \( A_j \) appears in parsing the strings in the sample set. (Maryanski and Booth 1977) then prove a theorem that the maximum likelihood estimate of the rule probability \( p_{i,j} \) is \( \frac{n_{i,j}}{N_{j}} \).

This is similar to the technique used in this research with the appropriate parallelism between the p-grammar structure and the PFSM structure. The rule set partitions correspond to the state set of the PFSM and their associated input symbol counts, \( I \) (as defined in
the notation chapter).

N corresponds to \( I \), which is the number of times \( J \) the input symbol (rule set partition) is observed while \( s \) is the current state (premise A). The count \( n \) \( J \) corresponds to the transition count, \( Q \), for the observed \( i-o \) pair for the state \( s \) (the count for rule \( r \) with the \( J \) premise A).

The transition probability in this research is estimated by \( Q/I \), which is \( n/N \) in (Maryanski and Booth 1977).

(Kountanis 1977) deals with the question of machine structure inference from the observation of the input-output sequence behavior of a machine. In that dissertation, a learning system was designed which infers (or learns) a finite state machine which is completely specified, connected and deterministic. The learning system converges to a unique machine in the class \( M(p,q,r) \) of machines if the input-output sequence consists of a "head" sequence which covers all the transitions of the machine in the black box and is followed by a "tail" sequence which distinguishes it from all other conjectured machines covered by the same "head"
sequence. The class $M(p, q, r)$ denotes the class of deterministic machines having the same input and output symbol sets, where $p$ is the cardinality of the input symbol set, $q$ is the cardinality of the output symbol set, and $r$ is an upper bound on the number of states of the machines in the class.

In this research, the learning strategy of (Kountanis 1977) is extended to cover the observed input/output behavior of a probabilistic finite state machine in the class $P(p, q, r)$ of all PFSM's with input symbol set cardinality $p$, output symbol set cardinality $q$, and an upper bound of $r$ on the number of states in the PFSM. $P(p, q, r)$ is essentially a probabilistic extension of $M(p, q, r)$.

(Biermann 1978) has developed a method for automatically creating programs from input - output examples of a class of regular automata. This work follows from previous developments of strategies of inferring a program structure by examining its traces.

In (Kountanis and Mitchell 1979), the complexity of the learning strategy of (Kountanis 1977) is further analyzed according to various representations of the learning strategy. Homomorphic images of the learning strategy are formed by dropping the transition labels

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
from the state transition digraph, then by combining transitions which have the same source and destination state, and finally by eliminating the direction from the transitions and forming a graph. Their results suggest that these homomorphic images of the learning strategy have a simpler problem space complexity, yet the space complexity remains enormous.
CHAPTER III

BACKGROUND: FUNDAMENTALS OF AUTOMATA THEORY

In this chapter, the fundamentals of automata theory will be reviewed. Different types of automata will be reviewed, starting with the finite automaton. After each type of automaton is defined, the next type of automaton will be introduced which contains an additional element of complexity. The automata that are reviewed are:

(1) deterministic finite automaton (dFA),
(2) nondeterministic finite automaton (nFA),
(3) deterministic finite state machines (DFSM),
(4) nondeterministic finite state machine (NDFSM),
and
(5) probabilistic finite state machine (PFSM).

A finite automaton (FA) is a 5-tuple
\[ M = \langle S, X, f, s_0, F \rangle \]

where

\( S \) is a finite set of internal states,
\( X \) is a non empty finite set of symbols called the input alphabet,
\( f \) is the state transition function,
s is the initial state of M, and
0
F is the set of final states, (F is a subset of S)

The state transition function of M is defined to be the mapping

\[ f: S \times X \rightarrow S \]

The finite automata, M, operates by transitioning to a new state \( s' \), upon the input of a symbol \( x \) from \( X \) while in state \( s \) of M, \((s, s', x) \) from S).

If, at the end of a sequence of input symbols, the state of the FA is a state in the set F, the FA is said to "accept" the sequence of input symbols.

The FA "accepts" the language, \( L \), which is defined to be the set of all the sequences of symbols from the input alphabet \( X \) which are accepted by the FA.

An example of the FA is the following machine which checks for odd parity on a sequence of binary digits:
\( M = < (s, s), (0, 1), f, s, \{s\} > \)

where \( f \) is defined to be

\[
\begin{align*}
(s, 0) &\rightarrow s \\
0 &\rightarrow 0 \\
(s, 1) &\rightarrow s \\
0 &\rightarrow 1 \\
(s, 0) &\rightarrow s \\
1 &\rightarrow 1 \\
(s, 1) &\rightarrow s \\
1 &\rightarrow 0
\end{align*}
\]

\( M \) may also be defined as a graph:

\[\text{Diagram of a graph with states } S_0 \text{ and } S_1, \text{ transitions between } 0 \text{ and } 1.\]
In addition, the FA may also be specified by using a next state table to specify the function $f$:

\[
\begin{array}{c|cc}
\text{present state} & 0 & 1 \\
\hline
0 & s & 0 \\
1 & s & 1 \\
\end{array}
\]

The FA just described is a deterministic FA, or dFA. Another form of the FA is the nondeterministic FA, or nFA.

The nFA is defined as is the dFA, with the exception of the next state transition function, $f$:

$f: S \times X \rightarrow 2^S$, where $2^S$ denotes the power set of $S$ (the set of all subsets of $S$)

Rather than the next state being a single state as in the dFA, the next state now is a subset of the state set, $S$. Hence the former has the name deterministic (or fixed), whereas the nondeterministic FA requires a choice in the next state transition.
Following the finite automaton in complexity is the finite state machine (FSM). Whereas the FA has the function of accepting a sequence of symbols, the FSM functions as a symbol transducer (or converter). The difference between the FA and the FSM will be the deletion of the final state set and the addition of an output symbol set, with a corresponding change to the transition function in order to output symbols.

A deterministic finite state machine (FSM) is a 5-tuple

\[ M = < S, X, Y, f, s_0 > \]

where

- \( S \) is a finite set of states,
- \( X \) is a finite set of input symbols,
- \( Y \) is a finite set of output symbols,
- \( f \) is the next-state/output transition function, and
- \( s \) is the initial state of \( M \)

The next-state/output transition function is defined to be the mapping

\[ f: S \times X \rightarrow S \times Y \]
An example of the DFSM is:

\[ M = \langle \{ s, s, s, s \}, \{ a, b, c \}, \{ 0, 1 \}, f, s \rangle \]

where \( f \) is defined as

<table>
<thead>
<tr>
<th>present</th>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>s /0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>s /1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>state</td>
<td>s</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>s</td>
<td>s /1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
The following is an example of the next state and output symbol sequence generated by this DFSM:

Input Sequence:  a b c a b c a b c a b c
Next State:  s s s s s s s s s s s s s s s s s s 2 4 4 3 3 4 3 3 4 3 3 4
Output Sequence:  0 0 1 1 1 0 1 1 0 1 1 0

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The deterministic finite state machine can be extended to a nondeterministic finite state machine in an analogous manner in which the dFA was extended to the nFA. The NDFSM is defined as is the DFSM, with the exception of the next-state/output transition function, \( f: \)

\[ f: S \times X \rightarrow 2^{(S \times Y)} \]

Once again, the next-state/output transition function requires a choice in the selection of the transition from some element of the power set of \( S \times Y \).

In the probabilistic finite state machine (PFSM), a probability is assigned to each of the nondeterministic transitions of a NDFSM:

\[ p_{i j k l} \] is the probability that when \( s \) is the current state, and the input symbol is \( x \), the symbol \( y \) is output and state \( s \) becomes the current state.
The next-state/output transition function may be specified as a matrix, $G$, whose rows correspond to the states of the PFSM, and columns correspond to the input symbols from the set $X$ of the PFSM:

$$
\begin{array}{cccc}
1 & 2 & \ldots & p \\
1 & 11 & 12 & 1p \\
2 & 21 & 22 & 2p \\
\vdots & \vdots & \vdots & \vdots \\
1 \quad 11 & 12 & 1p \\
\end{array}
$$

Each $g_{ik}$ in the matrix $G$ now is defined to be a $q$-element vector, with the element $1$ of $g_{ik}$ specifying the next state $s$ and the transition probability (with output symbol $y$). Note that the sum of all $q$ elements of $g_{ik}$ must equal 1.

Each column of the matrix $G$ may also be viewed as a matrix of $q$ columns, since each $g_{ik}$ has $q$ elements.
Thus, for the input symbol \( x \), the matrix \( H \) consisting of column \( k \) of \( G \) would be

\[
\begin{array}{cccc}
1 & 2 & \ldots & q \\
1 & 11 & 12 & 11 \\
2 & 21 & 22 & 2q \\
\vdots & \vdots & \vdots & \vdots \\
s & h & h & \ldots & h \\
s & h & h & \ldots & h \\
r & r_1 & r_2 & \ldots & rq \\
\end{array}
\]

where each element \( h \) would be the pair \( (s, p) \).

The deterministic FSM is a special case of the probabilistic FSM. In each row of all the \( H \) transition matrices for the PFSM, the column with the next state and output symbol for the deterministic transition will have a probability of 1. All other columns in each row have probabilities of 0. The DFSM
would be defined as the PFSM with the H matrices:

\begin{verbatim}
H:
\begin{array}{ccc}
& 0 & 1 \\
 s 1 & s /1.0 & --/0.0 \\
 s 2 & --/0.0 & s /1.0 \\
 s 3 & s /1.0 & --/0.0 \\
\end{array}
\end{verbatim}
which is equivalent to the next-state/output transition function table definition for the DFSM:
CHAPTER IV

PROBLEM SCOPE, METHODOLOGY, AND NOTATION

Scope

A learning strategy exists (Kountanis 1977) for the identification of the structure of a deterministic automaton from a class $M(p,q,r)$ of automata. An implementation of the deterministic learning strategy will be extended to determine the structure of a probabilistic automaton from a class of probabilistic automata with stationary transition probabilities. The class of probabilistic automata will be $P(p,q,r)$, where $P(p,q,r)$ has the same meaning as in the deterministic automaton case:

- $p$ is the cardinality of the input symbol set $X$,
- $q$ is the cardinality of the output symbol set $Y$, and
- $r$ is the upper bound on the number of states.

All of the probabilistic automata in the class $P(p,q,r)$ have the same input symbol set and output symbol set, namely, $X$ and $Y$, respectively. They also have the same upper bound, $r$, on the number of states. Another assumption is the existence of a reset function to set
the current state of the probabilistic automaton to its
initial state (multiple experiment).

Notation

At this point, the notation that will be used in the
remainder of this research will be defined.

Set cardinality: If A is a set, then \(|A|\) is the number
of elements (cardinality) of the set A.

S is the finite set of machine states of a machine,
X is the finite set of input symbols, and
Y is the finite set of output symbols.
S_{uv}, X_{uv}, and Y_{uv} denote the sets S, X, and Y,
respectively, for the uth machine at level v of the
learning strategy.

p = |X|, the cardinality of X for a class of FSM's,
q = |Y|, the cardinality of Y for a class of FSM's, and
r is an upper bound on the number of states in S for
machines in a class of FSM's.

M(p,q,r) is the class of deterministic FS machines with
p, q, and r defined as above.
P(p,q,r) is the class of probabilistic FS machines with
p, q, and r defined as above.
F is the total number of experiments.
E is the current experiment number

Z is an observed input-output sequence for an experiment

\[ Z \text{ is the observed i-o sequence for the current experiment } E \]

W is the length of an experiment (the number of observed input-output pairs that define the experiment sequence)

\[ W \text{ is the length of experiment } u \]

K is the current level of the learning strategy (the number of i-o pairs observed)

T is the next level of the learning strategy, \( T = K + 1 \)

C is the set of conjectured machines

\[ C \text{ is the set of conjectured machines at the current level, } K \]

\( N = |C| \)

\[ N = |C| \], the number of machines at the current level, \( K \), of the learning strategy

\( N \) is the number of machines with \( v \) states at level \( u \) of the learning strategy

m is a conjectured machine

\[ m = \text{uth machine at level } v \text{ of the learning strategy} \]
x is an input symbol from the set X
\[ x \in \text{the observed input symbol in the i-o pair} \]
\[ k \]

y is an output symbol from the set Y
\[ y \in \text{the observed output symbol in the i-o pair} \]
\[ l \]

d = current i-o pair subscript in the current experiment

f is the state transition/output function of a machine
\[ f \in f \text{ for the machine } m \]
\[ uv \]
\[ uv \]

s is the state u of a machine
\[ u \]

i = current state number
\[ s = \text{current state} \]
\[ i \]

J is a state count
\[ J = \text{number of times the current state was } s \]
\[ u \]

I is a state/symbol count
\[ I = \text{number of times the input symbol } x \text{ was observed while the current state was } s \]
\[ v \]
\[ u \]

Q is a transition count

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Methodology

In a deterministic FSM which is completely specified, every state of the machine must have a transition defined for each of the input symbols in X, but only one transition for each of these symbols in order to maintain determinism. For the class M(p,q,r), the completely specified state i must have exactly p transitions emanating from it. At each state of the DFSM, when given an input symbol, there is no choice of transition. One and only one transition for each input symbol exists at each state:
Note that the output symbol labelling is arbitrary, as is the destination state. In a completely specified DFSM, each state will have \( p \) output transitions (one for each input symbol). Each of these transitions can have one of \( q \) output symbol labellings.

One view of the probabilistic FSM is that it has the structure of a NDFSM with probabilities associated with each transition, with all the transitions that are nondeterministic for a symbol having probabilities such that their sum is 1. Nondeterminism is introduced when the same input symbol is labelled on more than one emanating transition. The NDFSM has transition choices.
given an input symbol. That is, a state may have emanating transitions with an input symbol labelling occurring on more than one transition.

There are several potential degrees (or types) of nondeterminism. These degrees or types may be viewed with respect to the parameters of any transition: the destination state, the output symbol labelling, or the input symbol labelling.
"State" nondeterminism: In this type of nondeterminism, the transition may be duplicated to another state with the same input and output symbol labelling requiring a choice in the next state of the machine:
"Output" nondeterminism: In this type of nondeterminism, the nondeterministic element of the transition is the output symbol labelling. The same transition may be duplicated with a change in the output symbol labelling requiring a choice in the occurrence of the output symbol:
"Input" nondeterminism: The nondeterministic element here is the input symbol labelling:

\[(x_{k1}, x_{k2}, \ldots) / y_1\]

However, this is identical to

\[x_{k1} / y_1 \quad x_{k2} / y_1 \quad \ldots\]
which is deterministic. This implies that "input" nondeterminism does not exist apart from "state" or "output" nondeterminism. Rightly so, since the nondeterminism, or multiple appearance of an input symbol labelling occurs because of more than one transition existing due to a "state" or "output" nondeterminism. A transition is nondeterministic because for a given input symbol \( x \), a choice in output exists:

\[ x_k^{k'}(y_1, y_2, \ldots) \]

\[ S_i \rightarrow S_j \]
or a choice in next state exists:

![Diagram](image)

or a combination of both exists:

![Diagram](image)
For a completely specified NDFSM, there exists transition arcs between all states such that a complete enumeration of all the possible input-output labels exists. For any two states $s_i$ and $s_j$, the complete enumeration is

$$
S_i \xrightarrow{x_1/(y_1, y_2, \ldots, y_q)} S_j \\
S_i \xrightarrow{x_2/(y_1, y_2, \ldots, y_q)} S_j \\
\vdots \\
S_i \xrightarrow{x_p/(y_1, y_2, \ldots, y_q)} S_j
$$
The PFSM which is modelled from the class \( P(p,q,r) \) of machines was defined in Chapter III. By examining the number of states in the \( G \) (and \( H \)) matrices of the PFSM, the number of possible transitions can be seen.

For any \( x \), the matrix \( H \) is an \( r \) by \( q \) matrix, defining the transition's output symbol, next state, and probability. Since each matrix \( H \) is a column in the matrix \( G \), which is an \( r \) by \( p \) matrix, there are a total of \( pqr \) transitions in the PFSM. This is less than the number of transitions for a completely specified NDFSM (which could have a probability assigned to each transition thereby becoming a completely specified PFSM) by a factor of \( r \). This factor of \( r \) is due to the next state being deterministic with respect to the output symbol (whereas the output symbol is nondeterministic with respect to the input symbol), otherwise another \( r-1 \) next states could also be chosen, increasing the complexity by \( r \).

However, if the next state was also allowed to be nondeterministic, the problem space of the learning strategy would become infinitely wide. Consider a completely specified PFSM \( m \) at level \( K \) of the learning.
strategy. Given the observed i-o pair \((x, y)\), and a configuration of \(m\) for the current state as \(z^K\)

At level \(K+1\), \(r\) machines will be generated from the machine \(m\) by repeating each of the \(r\) transitions \(z^K\) labelled \(x/y\). Since each of the machines generated from machine \(m\) are copies of \(m\), they will also be \(z^K\)
completely specified PFSM's and each will have r successor machines, leading to an infinitely wide problem space.

By restricting the next state to be deterministic, the observed i-o pair $x, y$ will only generate one machine from $m$ if there exists a transition already labelled $x / y$ from the current state $s$ in $m$.

The method employed in this research is to build upon the work of (Kountanis 1977) where a learning strategy for the class $M(p, q, r)$ of deterministic automata was designed. An implementation of that design is developed. Then the implementation for the deterministic case is extended in order to handle the input – output sequence of a probabilistic automaton over multiple experiments.
CHAPTER V

DETERMINISTIC LEARNING STRATEGY DESIGN
AND IMPLEMENTATION

The learning strategy of (Kountanis 1977) is represented as a semilattice of conjectural machines which cover the observed input - output behavior of a deterministic automaton. The learning strategy maintains a set of machines whose structures cover the input - output behavior of the automaton which has been observed by the learning strategy. As each machine is conjectured from its predecessor machine in the learning strategy, its current state is updated in the manner described below. As each new input - output pair of the automaton is observed, the learning strategy updates its set of conjectural machines by replacing each predecessor machine with the new machines which are inferred by the observed input - output pair, eliminating conjectural machines that contradict the observed behavior. It is essentially a pruning technique.

This process is carried out through the application of two rules to each of the conjectured machines presently at the memory of the learning strategy. The first rule attempts to cover the observed input - output behavior by introducing a new machine for each of the
existing states of the predecessor machine. In each of the successor machines, the current state becomes the state for which the successor machine was created. A transition to this state is added from the state which was the current state in the predecessor machine, and it is labelled with the input - output behavior.
If a transition already exists in the direction of this added transition which has the same input symbol labelling as the observed input symbol, the conjectured machine cannot be included if the output symbol labelling of the transition is not the same as the observed output symbol, since this would invalidate the determinism of the machine.
The second rules tries to cover the latest input-output behavior by introducing a new state in the successor machine, and labelling a transition from the current state to the new state with the input-output behavior that is observed. The new state then becomes the new current state of the successor machine. This rule cannot be applied if the new state would exceed the bound on the number of states in the class $M(p,q,r)$, or the condition of retaining the determinism of the automata is violated as stated in rule 1.
Figure 3: Trace From M(2,2,2)
Learning Strategy for Class $M(p, q, r)$ of Deterministic Automata

**RULE 1**

For each machine conjectured at the current level of the learning strategy, given that it has $n$ states, generate $n$ new machines, each of which is a copy of itself with a transition labelled with the observed i-o pair from it's current state to another state in the set of $n$ states. The transition destination state becomes the current state of the new machine.

**RULE 1 CONFLICT**

There may exist new machines generated at the new level which already have the observed input symbol as a label on a transition from the current state of the machine.

**RULE 1 RESOLUTION**

If the observed output symbol is the same as the output symbol label for the transition, and the transition is to the same state, then this is a repeat cycle in the observed i-o behavior (another occurrence of the same transition) and the machine may be included in the set of conjectured machines. Otherwise, do not include the new machine in the set of conjectured machines.
RULE 2

For each machine conjectured at the current level of the learning strategy, generate a new machine which is a copy of itself with a transition labelled with the observed i-o pair from its current state to a new state. The new state becomes the current state in the new machine.

RULE 2 CONFLICT

New state > r or a Rule 1 conflict

RULE 2 RESOLUTION

Do not include the new machine in the set of conjectured machines.
ALGORITHM L8DM to derive the set of conjectured deterministic finite state machines in the class $M(p,q,r)$ from the input-output sequence of length $u$:

$$(x_1 y_1) (x_1 y_1) \ldots (x_1 y_1)$$

$\begin{array}{cccc}
  k & l & k & l \\
  1 & 1 & 2 & 2 \\
\end{array}

u \ u

[Step 1] Initialize the level of the learning strategy:

$$K \leftarrow 0$$
$$d \leftarrow 0$$

[Step 2] Initialize the conjectured machine set with the null machine:

$$C \leftarrow \{ \text{m} \}, \text{where}$$

$$K \leftarrow 1K$$

$$m = \{ X, Y, S, f \},$$
$$1K$$

$$X = \{ \}, Y = \{ \}, f = \{ \},$$

$$S = \{ s \}, \text{and}$$

$$1$$

$$i \leftarrow -1, L \leftarrow 1$$
$$1K \quad 1K$$

[Step 3] REPEAT

[Step 4] Initialize the next level of the learning strategy for processing the next i-o pair in the observed i-o sequence:

$$T \leftarrow K + 1 \text{ (new level of the learning strategy)}$$

$$c \leftarrow 0 \quad \text{(machine count for level } T \text{)}$$

$$C \leftarrow \{ \} \text{ (conjectured machine set empty initially)}$$
\[ d \leftarrow d + 1 \quad (\text{next i-o pair}) \]

\[ X \leftarrow X \cup \{ x \}_d \quad (\text{add observed input symbol } \ k \ \text{to the input symbol set}) \]

\[ Y \leftarrow Y \cup \{ y \}_d \quad (\text{add observed output symbol } l \ \text{to the output symbol set}) \]

[Step 5] Process all machines in the previous level:

\[
\text{FOR } i \text{ from 1 to } N \text{ DO } K
\]

[Step 6] Generate all possible machines with a transition emanating from the current state \( s \) of \( m \):

\[
\text{FOR } v \text{ from 1 to } L \text{ DO } zK
\]

[Step 7] Check if the observed i-o pair is a repeated transition in \( m \):

\[
\text{IF } f \text{ contains the transition } zK
\]

\[
(s , x ) \rightarrow (s , y ), \quad \text{THEN } zK
\]

\[
(i , k ) \rightarrow (i , l ), \quad \text{THEN } zK d
\]

[Step 8] Update the machine count at level \( T \):

\[ c \leftarrow c + 1 \]

[Step 9] Make a copy of the machine:

\[ m \leftarrow m \]

\[ cT \leftarrow zK \]
[Step 10] Update the current state in the new machine

\[ m : \]
\[ cT \]
\[ i \leftarrow v \]
\[ cT \]

[Step 11] Add the new machine generated to the conjectured machine set for level \( T \):

\[ C \leftarrow C \cup \{ m \} \]
\[ T \]
\[ T \]

ELSE (step 7)

[Step 12] Check that no transition conflict exists in

\[ m : \]
\[ zK \]

IF \( f \) does not contain \( zK \)
\[ (s \rightarrow x \rightarrow y) \] on any Left Hand Side, THEN
\[ i \]
\[ k \]
\[ cT \]
\[ d \]

[Step 13] Update the machine count at level \( T \):

\[ c \leftarrow c + 1 \]

[Step 14] Make a copy of the machine:

\[ m \leftarrow m \]
\[ cT \]
\[ zK \]

[Step 15] Add the new transition:

\[ f \leftarrow f \cup \{ (s \rightarrow x \rightarrow y) \rightarrow (s \rightarrow x \rightarrow v \rightarrow y) \} \]
\[ cT \]
\[ cT \]
\[ i \]
\[ k \]
\[ v \]
\[ 1 \]
\[ cT \]
\[ d \]
\[ cT \]
\[ d \]
Step 16: Update the current state in the new machine

\[ m \leftarrow c_T \]

\[ i \leftarrow v \leftarrow c_T \]

Step 17: Add the new machine generated to the conjectured machine set for level \( T \):

\[ C \leftarrow C \cup \{ m \} \]

\[ T \leftarrow T \cup c_T \]

ENDIF (step 12)

ENDIF (step 7)

ENDFOR (step 6)

Step 18: Check if another state may be added in the machine \( m \):

\[ z_K \]

\[ IF r > L \] THEN

\[ z_K \]

Step 19: Check that no transition conflict exist:

\[ IF \text{ does not contain} \]

\[ (s, i, k) \text{ on any Left Hand Side} \] THEN

\[ c_T \]

\[ d \]

Step 20: Update the machine count at level \( T \):

\[ c \leftarrow c + 1 \]
[Step 21] Make a copy of the machine:

\[
\begin{align*}
m & \leftarrow m \\
\end{align*}
\]

[Step 22] Create a new state number:

\[
\begin{align*}
n & \leftarrow L + 1 \\
\end{align*}
\]

[Step 23] Add the new state to the set of states in \( m \):

\[
\begin{align*}
S & \leftarrow S \cup \{ s \} \\
L & \leftarrow n \\
\end{align*}
\]

[Step 24] Add a transition with the new state as destination to the transition function:

\[
\begin{align*}
f & \leftarrow f \cup \{ (s, x) \rightarrow (s, y) \} \\
\end{align*}
\]

[Step 25] Make the new state the current state of the machine \( m \):

\[
\begin{align*}
i & \leftarrow n \\
\end{align*}
\]

[Step 26] Add the new machine to the conjectured machine set for level \( T \):

\[
\begin{align*}
C & \leftarrow C \cup \{ m \} \\
\end{align*}
\]

ENDIF (step 19)
ENDIF (step 18)

ENDFOR (step 5)

[Step 27] Update the current level of the learning strategy:
K <- T

[Step 28] Repeat loop ends when the i-o sequence is all processed or the set of conjectured machines becomes empty:
UNTIL (step 3)
d = u
or
C = {}
K

END OF ALGORITHM

Appendix A contains an implementation of the learning strategy for the deterministic automaton. At each level of the learning strategy, a list is maintained of the conjectured machines which cover the behavior of the black box as observed from the input - output pairs. The list is of the form

\[
\begin{array}{cccccc}
M_1 & M_2 & \ldots & M_w & M_k & k
\end{array}
\]

where each M is of the form Mx, x being a new machine.\]
number in the set of machines generated. Example machine
numbers are M5 and M37. The attributes of each machine
are maintained by setting properties of these list
elements to specified values. There are three main
properties for each machine. Property CURRENT-STATE is
the state number of the current state of the machine.
Property STATE-COUNT is the total number of states of the
machine. Property STATE-LIST is a list of the machine
states of the machine. This list is of the form
\[ (Q_1, Q_2, \ldots, Q_w), \ w = L \]
where each Q is of the form Mz-Sv, v being the state
number of machine z. Examples are M5-S1, state 1 of
machine 5, and M37-S4, state 4 of machine 37. Each state
has a property named TRANSITIONS which describes the
state transition and output function of the machine for
that state. The value of the property TRANSITIONS is a
transition list of the form
\[ (T_1, T_2, \ldots, T_n) \]
Each T is a list of the form
\[ (\sigma, q, \delta ) \], where
sigma is an input symbol.

$q$ is the next state, and

delta is an output symbol.

Thus, the state list

\[( M_6-S1 \ M_6-S2 )\]

along with the transition lists

\[( (b \ 1 \ 1) \ (a \ 2 \ 0) ) \] for state $M_6-S1$

\[( (b \ 1 \ 0) \ (a \ 2 \ 1) ) \] for state $M_6-S2$

describes the following functions of machine $M_6$:

<table>
<thead>
<tr>
<th>NEXT-STATE</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>1</td>
<td>2/0</td>
</tr>
<tr>
<td>present</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2/1</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER VI

PROBABILISTIC LEARNING STRATEGY DESIGN AND IMPLEMENTATION

As in the case of the learning strategy for the deterministic automaton, the learning strategy for the probabilistic automaton creates a set which consists of conjectured machines which cover the observed input-output behavior of the probabilistic automaton under test. This learning strategy is based upon the learning strategy for deterministic automata, with extensions that provide for the computation of probabilities for the state transitions.

The learning strategy conjectures the set of machines for a single experiment consisting of an input-output sequence of finite length. The single experiment is repeated with the same input symbol sequence and the new output symbol sequence of the probabilistic automaton for a multiple experiment in order that the state transition probabilities of the conjectured machines converge to a common covering of the probabilistic automaton input-output behavior. This assumes that a reset of the probabilistic automaton is available before the execution of each single experiment.
This learning strategy for the probabilistic automaton consists of the rules which form the learning strategy for the deterministic automaton, plus the extension which is based upon a relaxing of the determinism constraint for the input symbol labelling of the state transitions of the conjectured machines. The extension allows for non-deterministic input symbol labellings along with the deterministic output symbol labellings of the state transitions of the conjectured machines. As may be expected, this learning strategy has a larger complexity than the learning strategy for the deterministic automata. This will be discussed in the next chapter. These non-deterministic input symbol labellings of the state transitions of the conjectured machines are counted in order to develop their probability distribution from the observed input–output behavior of the probabilistic automaton.

In the first rule that governs the addition of new transitions to existing states of the machines, this learning strategy allows a new transition that would have been in conflict in the deterministic case with respect to the input symbol label. The new transition is allowed if the output symbol labelling remains deterministic.
Figure 4: Partial Trace From P(2,2,2)
Algorithm for the Probabilistic Automaton Learning Strategy

RULE 1

For each machine conjectured at the current level of the learning strategy, given that it has n states, generate n new machines, each of which is a copy of itself with a transition labelled with the observed i-o pair from its current state to another state in the set of n states. The transition destination state becomes the current state of the new machine.

RULE 1 CONFLICT

There may exist new machines generated at the new level which already have the observed input symbol as a label on some transition from the current state of the machine.

RULE 1 RESOLUTION

If the observed output symbol is the same as the transition, and the transition is to the same state, then this is a repeat cycle in the observed i-o behavior (another occurrence of the same transition) and the machine may be included in the set of conjectured machines, after incrementing the transition count for the observed i-o pair transition. Otherwise, the output symbols do not match, and a new machine may be introduced with the new input - output labelling on the transition.
RULE 2

For each machine conjectured at the current level of the learning strategy, generate a new machine which is a copy of itself with a transition labelled with the observed i-o pair from its current state to a new state. The new state becomes the current state in the new machine.

RULE 2 CONFLICT

New state > r or a Rule 1 conflict

RULE 2 RESOLUTION

If the current state already has an input symbol label on an emanating transition, but is labelled with a different output symbol, introduce the new state and transition with a transition count of 1. Otherwise, do not include the new machine in the set of conjectured machines.
ALGORITHM LSPM to derive the set of conjectured probabilistic finite state machines in the class $P(p, q, r)$ from $F$ experiments

$$Z \quad Z \quad \ldots \quad Z$$

where each $Z$ is an input-output sequence of length $W$:

$$E \quad E$$

$$\begin{array}{cccc}
(x_1 y_1) & (x_1 y_1) & \ldots & (x_1 y_1) \\
1 & 1 & 2 & 2 & W & W & E & E
\end{array}$$

[Step 1] Initialize the learning strategy:

$$E \leftarrow 0 \quad \text{(experiment number)}$$

[Step 2] Initialize the level of the learning strategy:

$$K \leftarrow 0$$

[Step 3] Initialize the conjectured machine set with the null machine:

$$C \leftarrow \{ m \}, \text{ where}$$

$$K \leftarrow 1K$$

$$m = \{ X, Y, S, f \},$$

$$1K$$

$$X = \{ \}, \text{ } Y = \{ \}, \text{ } f = \{ \},$$

$$1K$$

$$S = \{ \varsigma \},$$

$$1$$

$$i \leftarrow -1, \text{ } L \leftarrow 1,$$

$$1K$$

$$j \leftarrow 0, \text{ and}$$

$$1K$$
I ← 0, u from 1 to p
1 1K

[Step 4] DO WHILE E < F

[Step 5] Perform the next experiment:
E ← E + 1
d ← 0 (i-o sequence number)
set s ← s for all machines
i 1 (reset all copies)

[Step 6] REPEAT

[Step 7] Initialize the next level of the learning strategy for processing the next i-o pair in the observed i-o sequence Z:
E
T ← K + 1 (new level of the learning strategy)
c ← 0 (machine count for level T)
C ← {} (conjectured machine set empty initially)
d ← d + 1 (next i-o pair)
X ← X U \{ x \} (add observed input symbol \( k \) to the input symbol set)
d
Y ← Y U \{ y \} (add observed output symbol \( l \) to the output symbol set)
d

[Step 8] Process all machines in the previous level:
FOR i from 1 to N DO
[Step 9] Generate all possible machines with a transition emanating from the current state $s$ of $m$:

$$\begin{align*}
&\text{FOR } v \text{ from } 1 \text{ to } L \text{ DO} \\
&\text{STEP 10 Check if the observed i-o pair is a repeated transition in } m:
\end{align*}$$

$$\begin{align*}
&\text{IF } f \text{ contains the transition } \\
&(s, x, t) \rightarrow (s, y) \text{ for any } t, \text{ THEN} \\
&\text{STEP 11 Update the machine count at level } T:
\end{align*}$$

$$\begin{align*}
&c \leftarrow c + 1
\end{align*}$$

[Step 12] Make a copy of the machine:

$$\begin{align*}
m &\leftarrow m \\
cT &\leftarrow T
\end{align*}$$

[Step 13] Update the current state count in $m$:

$$\begin{align*}
&J \leftarrow J + 1 \\
i &\leftarrow i \\
cT &\leftarrow cT
\end{align*}$$

[Step 14] Update the observed input symbol count for the current state of $m$:

$$\begin{align*}
&I \leftarrow I + 1 \\
i &\leftarrow i \\
cT &\leftarrow cT
\end{align*}$$
[Step 15] Update the transition count for the repeated transition:

\[ Q \leftarrow Q + 1 \]
\[ i \overset{v k}{\rightarrow} i \overset{v k}{\rightarrow} C_T d d \]

[Step 16] Update the current state in the new machine

\[ m : \]
\[ c_T \]
\[ i \leftarrow v \]
\[ c_T \]

[Step 17] Add the new machine generated to the conjectured machine set for level \( T \):

\[ C \leftarrow C \cup \{ m \} \quad T \quad c_T \]

ELSE (step 10)

[Step 18] Check that no transition conflict exists in

\[ m : \]
\[ z_K \]

IF \( f \) does not contain \( z_K \)

\( (s, x, t) \rightarrow (s, y) \), for any \( t, i, k \) and for all \( z_K d \)

THEN

[Step 19] Update the machine count at level \( T \):

\[ c \leftarrow c + 1 \]
[Step 20] Make a copy of the machine:

\[
\begin{align*}
\text{m} & \leftarrow \text{m} \\
\text{cT} & \leftarrow \text{zK}
\end{align*}
\]

[Step 21] Update the current state count in \text{m}:

\[
\begin{align*}
\text{m} & : \\
\text{cT} & \\
\text{J} & \leftarrow \text{J} + 1 \\
\text{i} & \leftarrow \text{i} \\
\text{cT} & \text{cT}
\end{align*}
\]

[Step 22] Update the observed input symbol count for the current state of \text{m}:

\[
\begin{align*}
\text{I} & \leftarrow \text{I} + 1 \\
\text{i} & \leftarrow \text{i} \\
\text{k} & \leftarrow \text{k} \\
\text{cT} & \text{d} \\
\text{cT} & \text{d}
\end{align*}
\]

[Step 23] Initialize the transition count for the new transition:

\[
\begin{align*}
\text{Q} & \leftarrow 1 \\
\text{i} & \leftarrow \text{v} \\
\text{k} & \leftarrow \text{k} \\
\text{v} & \leftarrow \text{v} \\
\text{i} & \leftarrow \text{i} \\
\text{cT} & \text{d} \\
\text{d} & \leftarrow \text{d}
\end{align*}
\]

[Step 24] Add the new transition:

\[
\begin{align*}
\text{f} & \leftarrow \text{f} \cup \{(s, x, t) \rightarrow (s, y)\} \\
\text{cT} & \text{cT} \\
\text{i} & \leftarrow \text{i} \\
\text{k} & \leftarrow \text{k} \\
\text{i} & \leftarrow \text{i} \\
\text{v} & \leftarrow \text{v} \\
\text{i} & \leftarrow \text{i} \\
\text{cT} & \text{d} \\
\text{d} & \leftarrow \text{d} \\
\text{cT} & \text{d} \\
\text{d} & \leftarrow \text{d}
\end{align*}
\]

[Step 25] Update the current state in the new machine:

\[
\begin{align*}
\text{m} & : \\
\text{cT} & \\
\text{i} & \leftarrow \text{v} \\
\text{cT}
\end{align*}
\]
[Step 26] Add the new machine generated to the conjectured machine set for level T:

\[ C \leftarrow C \cup \{ m \} \]

T \rightarrow T \quad c_T

ENDIF (step 18)

ENDIF (step 10)

ENDFOR (step 9)

[Step 27] Check if another state may be added in the machine m:

z_K

IF \( r > L \) THEN

z_K

[Step 28] Check that no transition conflict exist:

IF \( f \) does not contain

z_K

\((s, i, k, t) \rightarrow (s, w, l)\) for any \( t, i, k, w, l \) and for all

z_K \quad z_K \quad d \quad z_K \quad d \quad w \) in \([1, L] \)

THEN

[Step 29] Update the machine count at level T:

\[ c \leftarrow c + 1 \]

[Step 30] Make a copy of the machine:

\[ m \leftarrow m \]

\[ c_T \quad z_K \]
[Step 31] Create a new state number:

\[ n \leftarrow L + 1 \]

[Step 32] Initialize the state count for the new state in \( m \):

\[ cT \]

\[ J \leftarrow 0 \]

[Step 33] Initialize the observed input symbol counts for the new state in \( m \):

\[ cT \]

\[ I \leftarrow 0, v \text{ from } 1 \text{ to } p \]

[Step 34] Add the new state to the set of states in \( m \):

\[ cT \]

\[ S \leftarrow S \cup \{ s \} \]

\[ cT \]

\[ L \leftarrow n \]

[Step 35] Update the current state count in \( m \):

\[ cT \]

\[ J \leftarrow J + 1 \]
[Step 36] Update the observed input symbol count for the current state of $m$:

\[
I \leftarrow I + 1
\]

[Step 37] Initialize the transition count for the new transition:

\[
Q \leftarrow 1
\]

[Step 38] Add the new transition with the new state as destination to the transition function:

\[
f \leftarrow f \cup \{(s, x, t) \rightarrow (s, y)\}
\]

[Step 39] Make the new state the current state of the machine $m$:

\[
i \leftarrow n
\]

[Step 40] Add the new machine to the conjectured machine set for level $T$:

\[
C \leftarrow C \cup \{m\}
\]

ENDIF (step 28)

ENDIF (step 27)

ENDFOR (step 8)
[Step 41] Update the current level of the learning strategy:
K ← T

[Step 42] Repeat loop ends when the i-o sequence for experiment E is all processed, or the set of conjectured machines becomes empty:
UNTIL (step 6)
d = W E
or
C = {} K

[Step 43] If the set of conjectured machines becomes empty, discontinue the remaining experiments:
IF C = {} THEN E ← F K

ENDWHILE (step 4)

END OF ALGORITHM

Appendix B contains an implementation of the learning strategy for the probabilistic automaton. The implementation is an extension of the deterministic automaton learning strategy implementation. The extension mechanizes a counting of transition labellings in order to develop the transition probabilities from the observed input-output behavior of the probabilistic automaton. At each level of the learning strategy, a
list is maintained of the conjectured machines which cover the behavior of the black box as observed from the input-output pairs. The list is of the form

\[
\begin{array}{ccc}
M_1 & M_2 & \ldots & M_w \\
1 & 2 & \ldots & w \\
K & K & \ldots & K
\end{array}
\]

where each \( M \) is of the form \( M_x \), \( x \) being a new machine number in the set of machines generated, as in the implementation for the deterministic automata. The attributes of each machine are maintained by setting properties of these list elements to specified values. There are three main properties for each machine. Property CURRENT-STATE is the state number of the current state of the machine. Property STATE-COUNT is the total number of states of the machine. Property STATE-LIST is a list of the machine states of the machine. This list is of the form

\[
\begin{array}{ccc}
Q_1 & Q_2 & \ldots & Q_w \\
1 & 2 & \ldots & w \\
K & K & \ldots & K
\end{array}
\]

where each \( Q \) is of the form \( M_z-S_v \), \( v \) being the state number of machine \( z \). Each state has a property named TRANSITIONS which describes the state transition and output functions of the machine for that state. The value of the property TRANSITIONS is a transition list of
the form

\[( T \ T \ \ldots \ T )\]

Each \( T \) is a list of the form

\[ ( \text{sigma} \ \text{q} \ \text{delta} \ \text{count} ) \]

where

\( \text{sigma} \) is an input symbol,

\( \text{q} \) is the next state,

\( \text{delta} \) is an output symbol, and

\( \text{count} \) is the number of times this transition has occurred in the input-output sequence.

In addition, each state has a property named SYMBOL-COUNT which is the total number of occurrences of each symbol in the input sequence when the state was the current state. The value of the property SYMBOL-COUNT is a list of the form

\[ ( (x \ I ) \ (x \ I ) \ \ldots \ (x \ I ) ) \]

where

\( v \) is the state that this list is defined for,

\( x \) is the \( w \)th input symbol, and
\(I_vw\) is the input symbol count for state \(v\) and symbol \(x\).
CHAPTER VII

COMPLEXITY OF THE LEARNING STRATEGY

The number of machines generated at each successive level of the learning strategy by the algorithm LSPM characterizes the width of the problem space generated which will be used as a measure of the space complexity of the learning strategy. At each level $K$ of the learning strategy, the number of machines generated is

$$N = |C|_K$$

where $C$ is the set of conjectured $K$ machines at level $K$ of the learning strategy.

An illustration of this problem space is shown in figure 5.

The question to resolve in analyzing the width complexity is "How many machines, $N_K$, are generated at level $K$, when the problem space exhibits the worst case (maximum) rate of growth?"
In examining the algorithm LSPM, the number of successor machines that will be generated from each machine at the current level of the learning strategy is a function of the observed i-o pair $x, y$ and the existing transition labelling on all emanating transitions from their current state. This function
of each machine \( m \) can be stated as:
\[ z_K \]

IF \( (x, y) \) is already labelled, THEN
\[ z_K \]
1 machine is generated (a repeat transition)
ELSE (not labelled already)

IF \( L < r \), THEN
\[ z_K \]
\( L + 1 \) machines are generated
ELSE
\[ z_K \]
r machines are generated
ENDIF

Thus, the number of machines that are generated from the observed i-o pair \( x, y \) by LSPM as successors to any \( k \) machine \( m \) in the level \( K \) of the learning strategy is
\[ z_K \]
between 1 and \( r \) because of the following conditions:

(1) If the current state of \( m \) has a transition labelled
\[ z_K \]
with the observed i-o pair \( x, y \), only one machine
\[ z_K \]
will be generated, otherwise

(2) Successor machines of \( m \) will be generated with a
\[ z_K \]
new transition labelled \( x, y \). If \( m \) has \( r \) states,
\[ z_K \]
then \( r \) successor machines will be generated. If the number of states is \( L < r \), \( L \) machines with \( \leq r \) states will be generated and one machine will be generated with \( L + 1 \) states. This will occur until \( L = r \).

Thus, the maximum number of machines that will be generated by condition (2) above will be \( r \) machines with \( r \) states. This means an absolute upper limit on the worst case width of the problem space at level \( K \) cannot exceed \( r \) machines. The worst case width is expected to be less than \( r \), however, since the initial machine has one state rather than \( r \) states, so the problem space will not be a balanced tree of machines with branching factor of \( r \) at each machine generation node.

Note that due to condition (1) above, whenever a machine \( m \) with \( L \) states at level \( K \) of the learning strategy has a transition already labelled with the observed i-o pair \( x, y \) emanating from its current state, the problem space due to that machine will be pruned from \( \min(L + 1, r) \) successor machines to one.
successor machine. Therefore, the worst case growth on
the width of the problem space of LSPM will be when no
pruning due to condition (1) occurs. This worst case
behavior can occur only if no observed i-o pair is ever
repeated.

Consider the width complexity of LSPM for the class
\( P(\infty, \infty, r) \), where the input and output symbol sets are

\[
X = \{ x, x, \ldots \}, \text{ and} \quad X_1, X_2
\]

\[
Y = \{ y, y, \ldots \}, \text{ respectively,} \quad Y_1, Y_2
\]

and the input-output sequence to LSPM is formed from \( X \)
and \( Y \) above such that no i-o pair in the sequence is ever
repeated. An example of such a sequence is

\[
(x_1 y_1) (x_2 y_2) (x_3 y_3) \ldots \quad \text{where the } u \text{th i-o} \quad
1 \quad 2 \quad 3
\]

pair is \( (x_u y_u) \)

With this i-o sequence, the width complexity of the
problem space is worst case since the only limit on its
growth is \( r \), and all potential successors of each machine
will be generated by LSPM, since a new transition to
every state of a machine can be added. Condition (1)
ever holds because no repeat transitions can ever occur.
What does this problem space look like? It is appropriate to apply the homomorphic mappings of the hierarchy of representation for the learning strategy as developed in (Kountanis and Mitchell 1979). A trace of the learning strategy of P(∞,∞,2) with the sequence

\[(x, y) (x, y) (x, y)\]

\[1 \ 1 \ 2 \ 2 \ 3 \ 3\]

in the first level, D1, of this hierarchy is shown in Figure 6.

The homomorphic mapping from the first level of this representation hierarchy, D1, to the fourth level of the hierarchy, D4, is accomplished by reducing D1 to D2 by removing the transition labelings, then D2 is reduced to D3 by replacing all multiple arcs between states by a single arc, and finally by reducing D3 to D4 by removing direction from all the arcs. Thus, the nodes in the fourth level, D4, of the representation hierarchy are unlabelled graphs.

The previous example of P(∞,∞,2) in the representation hierarchy level D4 is shown in Figure 7.
Figure 6: Level D1 Trace of P(ω,ω,2) for (x_1 y_1)(x_2 y_2)(x_3 y_3)
DEFINITION:
A machine count trace of the learning strategy for $P(\infty, \infty, r)$ will be level D5 in the representation hierarchy, where the nodes in the representation at level D5 are formed by replacing each node at level D4 by the number of states in the machine which is at the node in level D4.

Thus the level D5 machine count trace of $P(\infty, \infty, 2)$ for the i-o sequence

\[(x\ y)\ (x\ y)\ (x\ y)\ ...\]

for various values of $r$ in $P(\infty, \infty, r)$ are:


\[ p(\infty, \infty, 1): \]

\[ \begin{array}{ccc}
& 1 & \\
1 & 1 & \text{.}
\end{array} \]

\[ p(\infty, \infty, 2): \]

\[ \begin{array}{ccc}
& 1 & \\
1 & 2 & 2 \\
1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & \text{.}
\end{array} \]
An interesting property of these traces is that the trace for each class \( P(\infty, \infty, u) \) is a sub-trace of the trace for every higher class \( P(\infty, \infty, v) \), where \( v > u \).

One use of this property is that the machine count for any level \( K \) of \( P(\infty, \infty, v) \), is the number of machines with \( v \) states at level \( K \) in \( P(\infty, \infty, v) \) plus the number of machines with \( v-1 \) states at level \( K \) in \( P(\infty, \infty, v-1) \) plus the number of machines with \( v-2 \) states at level \( K \) in \( P(\infty, \infty, v-2) \) plus ..., which is:

\[
N_K = \sum_{u=0}^{v-1} N_{K(v-u)}
\]

Thus, the difference between each class of \( P(\infty, \infty, v-1) \) at level \( K \) and \( P(\infty, \infty, v) \) at level \( K \) of the learning
strategy is \( N \), the number of machines with \( v \) states at level \( K \). Also, level \( K \) of any machine count trace in \( P(\infty, \infty, r) \) can have no machines with more than \( \min(r, K+1) \) states. This follows from Lemma 3.1 of (Kountanis 1977).

The question now is "How many machines have \( v \) states at level \( K \) in the learning strategy of \( P(\infty, \infty, r) \)?" For each machine with \( v \) states at level \( K-1 \), the learning strategy can generate one successor machine. Also, for each machine with \( v-1 \) states at level \( K-1 \), the learning strategy can generate one machine with \( v \) states. Thus,

\[
N = vN + N_{kv} \quad (K-1)v \quad (K-1)(v-1) \quad 00
\]

\[
N = 1 \quad 01
\]

The following table gives some values for \( 1 \leq K \leq 7 \) and \( 1 \leq v \leq 8 \).
Table 1: Values of $N_{Kr}$ states

<table>
<thead>
<tr>
<th>level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>15</td>
<td>25</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>31</td>
<td>90</td>
<td>65</td>
<td>15</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>63</td>
<td>301</td>
<td>350</td>
<td>140</td>
<td>21</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>127</td>
<td>966</td>
<td>1701</td>
<td>1050</td>
<td>266</td>
<td>28</td>
<td>1</td>
</tr>
</tbody>
</table>

The total number of machines at any level $K$ for $P(\infty, \infty, r)$ is then

$$N = \sum_{u=1}^{r} N_{Kr},$$

where $N_{Kr}$ is the number of machines with $u$ states at level $K$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The problem space forms the following hierarchy:

![Diagram of problem space hierarchy](image)

Figure 8: Problem Space Hierarchy of $P(\infty, \infty, r)$

The number of levels of the learning strategy generated by the algorithm LSPM until each machine generates only one successor machine (every observed i-o pair $x, y$ in the remainder of the input-output sequence is labelled on some transition emanating from...
the current state) characterizes the depth of the problem space which will be used as a measure of the time complexity of the learning strategy. This measures how long (input length) before the problem spaces reaches its worst case width.

Figure 9: LSPM Problem Space Complexity of P(\infty, \infty, r)

In section A of the problem space additional machines are being generated because the observed i-o is not existing
on any transition labelling. In section B of the problem space no additional machines are being generated because every observed i-o pair exists as a label for some transition from the current state of every machine.

The question to resolve in analyzing the depth complexity is "How many i-o pairs of an input - output sequence are required to reach the point where every state of every machine at level K of the learning strategy has transition labels for every i-o pair that can occur in the i-o sequence, given that the i-o sequence exhibits the worst-case occurrence of i-o pairs?" This worst-case occurrence is such that it maximizes the length of the i-o sequence to K.

When no more additional machines are generated because every i-o pair is labelled on some transition, the occurrence of the i-o pairs further contribute to the convergence of the transition probabilities of the conjectured machines.

When all the completely specified machines in the class P(p,q,r) have been generated by LSPM, no more machines can be generated and the width will not increase. It was shown before that the completely specified machine in P(p,q,r) will have all output symbol labellings for each input symbol labelling. How many
completely specified machines, then, are in $P(p,q,r)$?

Since the machine must be completely specified, it follows that $pq$ transitions emanate from each state. Each one of the $pq$ transitions can have up to $r$ possible destinations. Thus,

transition label $x_1, y_1$ will have $r$ possible destinations,

transition label $x_1, y_2$ will have $r$ possible destinations,

transition label $x_p, y_q$ will have $r$ possible destinations.

or

$$
\prod_{i=1}^{r} pq = r^{pq} \text{ possible transitions for any state.}
$$

Likewise, each of these $r$ states have $r^{pq}$ possible transitions.

state $s_1$ has $r^{pq}$ possible transitions,

state $s_2$ has $r^{pq}$ possible transitions,

state $s_r$ has $r^{pq}$ possible transitions.
or \[ \prod_{r}^{pq} = \frac{pq}{r} \text{ possible machines} \]

The maximum length to completely specify all machines in \( P(p, q, r) \) is

\[ \frac{pq}{r} \]

\[ \frac{r}{r!} \]

since the state assignment is unique up to a reordering of the state names.
CHAPTER VIII

CONCLUSIONS AND RECOMMENDATIONS

In this research, an implementation of a learning strategy for the class \( M(p,q,r) \) of deterministic automata was accomplished. This learning strategy was then extended to an analogous class \( P(p,q,r) \) of probabilistic automata, and also implemented. The width and depth complexities of the learning strategy for the probabilistic automata were characterized and evaluated.

There are several recommendations to be made for further research in the inference of probabilistic automata. In this research, a Mealy model of the probabilistic FSM was used. What is the effect on the algorithm and complexity of the learning strategy if the black box to be inferred was modelled as a Moore model of a probabilistic automata? Are there heuristics that could be applied as a result of using a Moore model that would effectively reduce the complexity of the learning strategy?

A follow up step to this research needs to be the application of heuristics to aid in reducing the complexity of the problem space. These heuristics should have the goal of eliminating conjectural machines at each
level of the learning strategy. In conjunction with the implementation of heuristics, adding a teaching strategy is also recommended in order to aid the pruning process. Another related problem to investigate is the implementation of an appropriate teaching strategy to drive the learning strategies for these automaton classes.

Other heuristics that need investigation for this learning strategy are methods to determine the convergence of the transition probabilities to those of the black box which is being modelled as a PFSM. Indeed, if one had an a-priori knowledge of the nature of the transition probabilities in the black box, this could be integrated into the learning strategy. Other aspects that should be addressed would be the relationship between the pruning of the problem space by eliminating conjectural machines according to some heuristics and how to combine existing machines, given their individual transition probability distributions.

The multiple experiment without reset of the probabilistic automaton is very important to accomplish, since the reset is not available in many realistic environments. Also, with incorporating a teaching strategy, the investigation of adaptive experiments can
be addressed, instead of using just preset experiments as in this research. More investigation of the learning strategy for probabilistic automata is suggested by considering the learning of a system of inter-connected probabilistic automata. This system may be connected in a hierarchy. Given the individual probabilistic automata that comprise the hierarchy, and their input-output behavior in a multiple experiment, can the structure of the hierarchy be conjectured to be a single probabilistic automaton? This would seem to be a natural follow-up to this research, since the states of the PFSM may not be fully decomposed. By implementing probabilistic structures of the class \(P(p,q,r)\) of PFSM, this research has allowed the case that while the learning strategy is inferring the conjectural machines with up to \(r\) states that cover the observed input-output behavior, it may indeed be the case that the black box has more than \(r\) states. This method does not have the sophistication to recognize the situation that the black box model needs to be increased past \(r\) states, which necessitates the decomposition of states in the conjectured machines of the learning strategy.
An application of this learning strategy is the development of a disease model for use in medical diagnosis and suggested treatment. With a data base that consists of a sample of a diseased population whose disease symptoms, treatment and test results are recorded as a series of case histories, the input - output behavior of the disease is available for consideration as a probabilistic automaton whose structure is to be inferred. In order to treat the patient afflicted with the disease, the practitioner must have conceptualized a model of the structure of the disease. The practitioner has a degree of belief (i.e., a probability measure) that a particular treatment will cause the patient to transition from one disease state to another disease state. These probability measures are based upon experience with the diseased population. The model representation that is developed can be used in a consulting role as an expert knowledge base. A disease model can also be used as an aid in the appropriate environments as a tool for suggesting diagnosis or treatment for making medical decision procedures. In addition, the model may serve a useful function as an educational device for medical personnel in their training programs.
APPENDIX A

LEARNING STRATEGY IMPLEMENTATION FOR DETERMINISTIC AUTOMATA

This appendix contains the implementation of the learning strategy described in Chapter V for the class M(p,q,r) of deterministic automata. It was implemented in Interlisp-VAX on a VAX-11/780. The calling path structure of the program is shown, followed by the Lisp function definitions, and an example execution of the program.
1. TEST-LSM: BLACK-BOX
  2. LIST PRINT-RESULTS PRINT-MACHINES PRINT-MACHINE-CHANGE GET-CURRENT-STATE
  3. | PRINT-MACHINES (2)
  4. INITIAL-MACHINE PUT-STATE-COUNT
  5. | PUT-CURRENT-STATE
  6. | PUT-STATE-LIST
  7. | PUT-TRANSITION-LIST
  8. | MACHINE-NAME
  9. | STATE-NAME
 10. GET-OBSERVED-ID-PAIR
 11. NEW-MACHINE-LIST CREATE-DESCENDANT-MACHINES NEW-STATE-OK? GET-TRANSITION-LIST
 12. | | STATE-NAME
 13. | | GET-CURRENT-STATE
 14. | | GET-STATE-COUNT
 15. | | RULE1-MACHINES TRANSITION-CONFLICT? GET-TRANSITION-LIST
 16. | | | STATE-NAME
 17. | | | GET-CURRENT-STATE
 18. | | | NEW-TRANSITION PUT-TRANSITION-LIST
 19. | | | STATE-NAME
 20. | | | GET-CURRENT-STATE
 21. | | | GET-TRANSITION-LIST
 22. | | | PUT-CURRENT-STATE
 23. | | | GET-STATE-COUNT
 24. | | | CREATE-MACHINE PUT-STATE-COUNT
 25. | | | PUT-CURRENT-STATE
 26. | | | PUT-STATE-LIST
 27. | | | TRANSFER-STATE-DEFINITIONS (a)
 28. | | | MACHINE-NAME
 29. | | | GET-STATE-COUNT
 30. | | | GET-CURRENT-STATE
 31. | | | GET-STATE-LIST
 32. | | RULE2-MACHINES NEW-TRANSITION (b)
 33. | | PUT-STATE-LIST
 34. | | PUT-STATE-COUNT
 35. | | PUT-CURRENT-STATE
 36. | | CREATE-MACHINE (24)
 37. | | GET-STATE-COUNT
 38. | | STATE-NAME
 39. | | GET-STATE-LIST
 40. NEW-MACHINE-LIST (11)

41. TRANSFER-STATE-DEFINITIONS PUT-TRANSITION-LIST
 42. | PUT-STATE-LIST
 43. | GET-STATE-LIST
 44. | GET-STATE-COUNT
 45. | STATE-NAME
 46. | GET-TRANSITION-LIST
NIL

---

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(TEST-LSHM)

(ELAMBDA (R))

(PROGN (SETQ MAXIMUM-STATES R)
  (BLACK-BOX)
  (LSHM))

(EDITED:
  '13-NOV-82 13:55')
(progn
  (setq observed-io-behavior nil)
  (terpri)
  loop(print1 "Enter an i-o pair, or NIL when done! ")
  (setq pair (read))
  (cond
   ((not (null pair))
    (setq observed-io-behavior (append observed-io-behavior (list pair)))
   (go loop))
  (t nil))

(black-box)
(PROG (MACHINE-LIST IO-PAIR OBSINP OBSOUT)
  (SETO MACHINE-LIST (LIST (INITIAL-MACHINE)))
  LOOP (SETO IO-PAIR (GET-OBSERVED-IO-PAIR))
  (COND
    ((NULL IO-PAIR)
     NIL)
    (T (SETO OBSINP (CAR IO-PAIR))
     (SETO OBSOUT (CDR IO-PAIR))
     (SETO MACHINE-LIST (NEW-MACHINE-LIST MACHINE-LIST OBSINP OBSOUT))
     (PRINT-RESULTS MACHINE-LIST OBSINP OBSOUT)
     (GO LOOP)))
(LSDM)

-PP#(LSDM)

(LSDM
  (LAMBDA NIL
   (PROG (MACHINE-LIST IO-PAIR OBSINP OBSOUT)
     (SETO MACHINE-LIST (LIST (INITIAL-MACHINE)))
     LOOP (SETO IO-PAIR (GET-OBSERVED-IO-PAIR))
     (COND
      ((NULL IO-PAIR)
       NIL)
      (T (SETO OBSINP (CAR IO-PAIR))
       (SETO OBSOUT (CDR IO-PAIR))
       (SETO MACHINE-LIST (NEW-MACHINE-LIST MACHINE-LIST OBSINP OBSOUT))
       (PRINT-RESULTS MACHINE-LIST OBSINP OBSOUT)
       (GO LOOP)))
  )

# edited:
  '13-NOV-82 13:33'
(PRINT-RESULTS)

(PRINT-RESULTS
  (CLAMEDA (M-LIST INSYM OUTSYM)

  (PROGN (TERFRI)
  (PRINT "The machines conjectured from the observed input")
  (PRINT INSYM)
  (PRINT "and observed output")
  (PRINT OUTSYM)
  (PRINT "are:")
  (TERFRI)
  (TERFRI)
  (PRINT-MACHINES M-LIST)
  (TERFRI))

(PRINT-RESULTS)
(PRINT-MACHINE-CHANGE)

(PRINT-MACHINE-CHANGE)

(lambda (machine)

(print "Machine")

(print (pack (cdr (unpack machine))

(print "is conjectured from Machine")

(print (pack (cdr (unpack (getprop machine (quote parent-machine)))

(print "with a transition from")

(print (getprop machine (quote previous-state)))

(print "to")

(print (get-current-state machine))

(terpri)

(print-machine-change)

-

())))

(lambda (m-list)

(progn (print-machine-change (car m-list))

(cond

((null (cdr m-list))

nil)

(t (print-machines (cdr m-list)))

(print-machines)

-)
(PRINT (INITIAL-MACHINE)
(INITIAL-MACHINE
   (LAMBDA NIL)
(FIRST (MACHINE S-NAME))
   (setq MACHINE (MACHINE-NAME NIL))
   (PUT-STATE-COUNT MACHINE (QUOTE 1))
   (PUT-CURRENT-STATE MACHINE (QUOTE 1))
   (setq S-NAME (STATE-NAME MACHINE (QUOTE 1))
   (PUT-STATE-LIST MACHINE (LIST S-NAME))
   (PUT-TRANSITION-LIST S-NAME NIL)
   (RETURN MACHINE)
(INITIAL-MACHINE)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(PUT-CURRENT-STATE
  (PUTPROP MACHINE (QUOTE CURRENT-STATE)
    STATE3))

(PUT-STATE-COUNT
  (PUTPROP MACHINE (QUOTE STATE-COUNT)
    COUNT3))

(PUT-STATE-LIST
  (PUTPROP MACHINE (QUOTE STATE-LIST)
    S-LIST3))

(PUT-TRANSITION-LIST
  (PUTPROP STATE-ATOM (QUOTE TRANSITIONS)
    T-LIST3))
(GET-CURRENT-STATE)

(GET-CURRENT-STATE
  CLAMDA (MACHINE)
  (** edited:
    '11-NOV-82 10:16')

  (GETPROP MACHINE (QUOTE CURRENT-STATE))

  (GET-CURRENT-STATE)
  PP* (GET-OBSERVED-IO-PAIR)

  (GET-OBSERVED-IO-PAIR
   CLAMDA NIL
   (** edited:
    '13-NOV-82 13:37')

  (PROG (PAIR)
    (SETO PAIR (CAR OBSERVED-IO-BEHAVIOR))
    (SETO OBSERVED-IO-BEHAVIOR (CDR OBSERVED-IO-BEHAVIOR))
    (RETURN PAIR))

  (GET-OBSERVED-IO-PAIR)
  PP* (GET-STATE-COUNT)

  (GET-STATE-COUNT
   CLAMDA (MACHINE)
   (** edited:
    '11-NOV-82 10:48')

  (GETPROP MACHINE (QUOTE STATE-COUNT))

  (GET-STATE-COUNT)
  PP* (GET-STATE-LIST)

  (GET-STATE-LIST
   CLAMDA (MACHINE)
   (** edited:
    '12-NOV-82 13:16')

  (GETPROP MACHINE (QUOTE STATE-LIST))

  (GET-STATE-LIST)
  PP* (GET-TRANSITION-LIST)

  (GET-TRANSITION-LIST
   CLAMDA (STATE-ATOM)
   (** edited:
    '12-NOV-82 13:17')

  (GETPROP STATE-ATOM (QUOTE TRANSITIONS))

  (GET-TRANSITION-LIST)
  -
="PF*(MACHINE-NAME)

(MACHINE-NAME
  ELAMBDA NIL

  (GENSYM (QUOTE MJ))
(MACHINE-NAME)
="PF*(NEW-MACHINELIST)

(NEW-MACHINE-LIST
  ELAMBDA (M-LIST INSYM OUTSYM)

  (APPEND (CREATE-DESCENDANT-MACHINES (CAR M-LIST)
    INSYM OUTSYM)

  (COND
    ((NULL (CDR M-LIST))
      NIL)
    (T (NEW-MACHINE-LIST (CDR M-LIST)
      INSYM OUTSYM)))

(NEW-MACHINE-LIST)

"
(CREATE-DESCENDANT-MACHINES)

(CLAMBDA (MACHINE INSYM OUTSYM)

(APPEND (RULE1-MACHINES MACHINE INSYM OUTSYM)

(COND
  ((NEW-STATE-OK? MACHINE INSYM)
   (LIST (RULE2-MACHINE MACHINE INSYM OUTSYM)))

(CREATE-DESCENDANT-MACHINES)
(NEW-STATE-OK?)
  (lambda (machine insym)
      (cond
         ((null (assoc insym (get-transition-list (state-name machine)
                                    (get-current-state machine)))
          (igreaterp maximum-states (get-state-count machine))
          (t nil))
      (new-state-ok?)
      (pp*(transition-conflict?)
      (transition-conflict?)
      (lambda (machine state insym outsym)
         (prog (t-list s-list)
            (setq t-list (get-transition-list (state-name machine)
                                             (get-current-state machine))
            (setq s-list (assoc insym t-list))
            (cond
              ((null s-list)
               (return nil))
              ((equal (cadr s-list)
                      state)
               (return (not (equal outsym (caddr s-list))
                          (t (return t))
               (transition-conflict?)
               (return nil))
         (return nil))
         (return nil))
      (return nil)))

RULE1-MACHINES

(FOR (STATE NEW-MACHINE NEW-LIST)
  [for STATE from 1 to (GET-STATE-COUNT MACHINE)
    do (COND
      ((NOT (TRANSITION-CONFLICT? MACHINE STATE INSYM OUTSYM))
        (NEW-TRANSITION NEW-MACHINE STATE INSYM OUTSYM)
        (PUT-CURRENT-STATE NEW-MACHINE STATE)
        (PUT-STATE-LIST NEW-MACHINE (APPEND (GET-STATE-LIST NEW-MACHINE) (LIST STATE))
        (RETURN NEW-LIST))
      (RULE1-MACHINES)
    )]
  )

RULE2-MACHINE

(FOR (NEW-MACHINE STATE NSN)
  (NEW-MACHINE (CREATE-MACHINE MACHINE))
  (STATE (ADD1 (GET-STATE-COUNT MACHINE))
  (STATE-NAME NEW-MACHINE STATE)
  (NEW-TRANSITION NEW-MACHINE STATE INSYM OUTSYM)
  (PUT-STATE-LIST NEW-MACHINE (APPEND (GET-STATE-LIST NEW-MACHINE) (LIST STATE))
  (RETURN NEW-LIST))
  )

RULE2-MACHINE
(CREATE-MACHINE
 [LAMBDA (BASE-MACHINE) (* edited)

  '13-MOV-02 13:11')

(DEFINE (NEXT-MACHINE)
  (SETQ NEXT-MACHINE (MACHINE-NAME NIL))
  (PUT-STATE-COUNT NEXT-MACHINE (GET-STATE-COUNT BASE-MACHINE))
  (PUT-CURRENT-STATE NEXT-MACHINE (GET-CURRENT-STATE BASE-MACHINE))
  (PUT-STATE-LIST NEXT-MACHINE (GET-STATE-LIST BASE-MACHINE))
  (TRANSFER-STATE-DEFINITIONS BASE-MACHINE NEXT-MACHINE)
  (PUTPROP NEXT-MACHINE (QUOTE PREVIOUS-STATE)
    (GET-CURRENT-STATE NEXT-MACHINE))
  (PUTPROP NEXT-MACHINE (QUOTE PARENT-MACHINE)
    BASE-MACHINE)
  (RETURN NEXT-MACHINE))

(DEFINE-MACHINE)
(NEW-TRANSITION)

(lambda (machine state insym outsym)
  (prog (n-tran s-name)
    (setq n-tran (list insym state outsym))
    (setq s-name (state-name machine (get-current-state machine)))
    (cond
      ((member n-tran (get-transition-list s-name)) nil)
      (t (put-transition-list s-name (append (get-transition-list s-name)
        (list n-tran))
        (putprop machine (quote last-transition)
          n-tran)
        (return n-tran))
    (new-transition)
  )
)

-Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[\text{PP*(STATE-NAME)}\]

\[(\text{STATE-NAME})\]
\[\text{CLAMBDA (MACHINE STATE)}\]
\[\text{(PACK (APPEND (UNPACK MACHINE))}\]
\[\text{ (UNPACK (QUOTE -S))}\]
\[\text{ (UNPACK STATE))}\]
\[\text{PP*(TRANSFER-STATE-DEFINITIONS)}\]

\[(\text{TRANSFER-STATE-DEFINITIONS)}\]
\[\text{CLAMBDA (OLD-MACHINE NEW-MACHINE)}\]
\[\text{(PROG (OLD-S-LIST NEW-S-LIST STATE OSN NSN)}\]
\[\text{ (SETQ OLD-S-LIST (GET-STATE-LIST OLD-MACHINE))}\]
\[\text{ (SETQ NEW-S-LIST OLD-S-LIST)}\]
\[\text{ [for STATE from 1 to (GET-STATE-COUNT OLD-MACHINE)}\]
\[\text{ do ((SETQ OSN (STATE-NAME OLD-MACHINE STATE))}\]
\[\text{ (COND}\]
\[\text{ ((NOT (NULL OSN)))}\]
\[\text{ (SETQ NSN (STATE-NAME NEW-MACHINE STATE))}\]
\[\text{ (SETQ NEW-S-LIST (SUBST NSN OSN NEW-S-LIST))}\]
\[\text{ (PUT-TRANSITION-LIST NSN (GET-TRANSITION-LIST OSN)}\]
\[\text{ (PUT-STATE-LIST NEW-MACHINE NEW-S-LIST)}\]
\[\text{ (TRANSFER-STATE-DEFINITIONS)}\]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
_TEST-LSH(2)

Enter an i-o pair, or NIL when done! (A 0)
Enter an i-o pair, or NIL when done! (B 1)
Enter an i-o pair, or NIL when done! (A 1)
Enter an i-o pair, or NIL when done! (B 0)
Enter an i-o pair, or NIL when done! (A 0)
Enter an i-o pair, or NIL when done! ()

The machines conjectured from the observed input A and observed output 0 are:

Machine 6 is conjectured from Machine 5 with a transition from 1 to 1
Machine 7 is conjectured from Machine 5 with a transition from 1 to 2

The machines conjectured from the observed input B and observed output 1 are:

Machine 8 is conjectured from Machine 6 with a transition from 1 to 1
Machine 9 is conjectured from Machine 6 with a transition from 1 to 2
Machine 10 is conjectured from Machine 7 with a transition from 2 to 1
Machine 11 is conjectured from Machine 7 with a transition from 2 to 2

The machines conjectured from the observed input A and observed output 1 are:

Machine 12 is conjectured from Machine 9 with a transition from 2 to 1
Machine 13 is conjectured from Machine 9 with a transition from 2 to 2
Machine 14 is conjectured from Machine 11 with a transition from 2 to 1
Machine 15 is conjectured from Machine 11 with a transition from 2 to 2

The machines conjectured from the observed input B and observed output 0 are:

Machine 16 is conjectured from Machine 13 with a transition from 2 to 1
Machine 17 is conjectured from Machine 13 with a transition from 2 to 2
Machine 18 is conjectured from Machine 14 with a transition from 1 to 1
Machine 19 is conjectured from Machine 14 with a transition from 1 to 2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The machines conjectured from the observed input B and observed output 0 are:

Machine 20 is conjectured from Machine 17 with a transition from 2 to 2
Machine 21 is conjectured from Machine 18 with a transition from 1 to 1

The machines conjectured from the observed input A and observed output 0 are:

Machine 22 is conjectured from Machine 21 with a transition from 1 to 2

NIL
-GET-STATE-LIST(M0022)
(M0022-S1 M0022-S2)
-GET-TRANSITION-LIST(M0022-S1)
((A 2 0) (B 1 0))
-GET-TRANSITION-LIST(M0022-S2)
((B 2 1) (A 1 1))

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX B

LEARNING STRATEGY IMPLEMENTATION FOR PROBABILISTIC AUTOMATA

This appendix contains the implementation of the learning strategy described in Chapter VI for the class $P(p, q, r)$ of probabilistic automata. It was implemented in Interlisp-VAX on a VAX-11/780. The calling path structure of the program is shown, followed by the Lisp function definitions, and an example execution of the program.
1. TEST-LSM: BLACK-BOX

2. LSPM PRINT-RESULTS PRINT-MACHINES PRINT-MACHINE-CHANGE GET-CURRENT-STATE
3. PRINT-MACHINES (2)

4. INITIAL-MACHINE PUT-STATE-COUNT
5. I PUT-CURRENT-STATE
6. I PUT-STATE-LIST
7. I PUT-TRANSITION-LIST
8. I MACHINE-NAME
9. I STATE-NAME
10. PUT-CURRENT-STATE
11. GET-DESCENDENT-TO-PAIR
12. NEW-MACHINE-LIST CREATE-DESCENDANT-MACHINES NEW-STATE-OK? OUTPUT-HERE?
13. PUT-STATE-COUNT
14. PUT-ALL-TRANSITIONS
15. PUT-TRANSITION-LIST
16. STATE-NAME
17. GET-CURRENT-STATE
18. RULE-MACHINES TRANSITION-CONFLICT OUTPUT-HERE?
19. STATE-NAME
20. GET-ALL-TRANSITIONS
21. GET-TRANSITION-LIST
22. STATE-NAME
23. GET-CURRENT-STATE
24. NEW-TRANSITION UPDATE-TRANSITIONS (a)
25. STATE-NAME
26. GET-CURRENT-STATE
27. PUT-CURRENT-STATE
28. GET-STATE-COUNT
29. CREATE-MACHINE PUT-STATE-COUNT
30. PUT-CURRENT-STATE
31. PUT-STATE-LIST
32. TRANSFER-STATE-DEFINITIONS (b)
33. MACHINE-NAME
34. GET-STATE-COUNT
35. GET-CURRENT-STATE
36. GET-STATE-LIST
37. RULE2-MACHINE NEW-TRANSITION (24)
38. PUT-STATE-LIST
39. PUT-STATE-COUNT
40. PUT-CURRENT-STATE
41. CREATE-MACHINE (29)
42. GET-STATE-COUNT
43. STATE-NAME
44. GET-STATE-LIST
45. NEW-MACHINE-LIST (12)

46. UPDATE-TRANSITIONS
47. PUT-TRANSITION-LIST
48. PUT-SYMBOL-COUNT PUT-SYMBOL-LIST
49. STATE-NAME
50. GET-SYMBOL-LIST
51. GET-SYMBOL-COUNT GET-SYMBOL-LIST
52. GET-TRANSITION-LIST
53. GET-ALL-TRANSITIONS
54. GET-SYMBOL-COUNT (49)

55. TRANSFER-STATE-DEFINITIONS
56. PUT-TRANSITION-LIST
57. PUT-SYMBOL-LIST
58. PUT-STATE-LIST
59. STATE-NAME
60. GET-STATE-LIST
61. GET-TRANSITION-LIST
62. GET-SYMBOL-LIST
NIL
(FP#(TEST-LSFH)

(TEST-LSFH
  (LAMBDA (R)
    (PROGN (SETO MAXIMUM-STATES R)
      (BLACK-BOX)
      (LSFH)))

(TEST-LSFH)

(* edited: 
  "28-NOV-82 16:02")
(PP*(BLACK-BOX)

(BLACK-BOX
 (CLAMBDA NIL
   (PROG (SEQ)
     (SETQ EXPERIMENT-LIST NIL)
     (TERPRI)
     LOOP(PRINT 'Enter an i-o experimenter, NIL when done: ')
     (SETQ SEQ (READ))
     (COND
      ((NOT (NULL SEQ))
       (SETQ EXPERIMENT-LIST (APPEND EXPERIMENT-LIST (LIST SEQ)))
       (GO LOOP))
      (T NIL))
   (BLACK-BOX)
   )
  )}
(LSFM)

(LSPM)

[LAMBDA NIL]

(PROG (MACHINE-LIST IO-PAIR OBSINF OBSOUT)
    (SETO MACHINE-LIST (LIST (INITIAL-MACHINE)))
    (SETO EXPERIMENT 0)
    OLOOP
      (SETO EXPERIMENT (ADD1 EXPERIMENT))
      (SETO OBSERVED-ID-BEHAVIOR (CAR EXPERIMENT-LIST))
      (MAPCAR MACHINE-LIST (QUOTE (LAMBDA (M))
        (PUT-CURRENT-STATE M 1))
      (TERPRI)
      (PRIN1 "EXPERIMENT ")
      (PRIN1 EXPERIMENT)
      (PRIN1 "i ")
      (PRIN1 OBSERVED-ID-BEHAVIOR)
      (TERPRI)
    ILOOP
      (SETO IO-PAIR (GET-OBSERVED-ID-PAIR))
      (COND
        ((NULL IO-PAIR)
          NIL)
        (T (SETO OBSINF (CAR IO-PAIR))
            (SETO OBSOUT (CDR IO-PAIR))
            (SETO MACHINE-LIST (NEW-MACHINE-LIST MACHINE-LIST OBSINF OBSOUT))
            (PRINT-RESULTS MACHINE-LIST OBSINF OBSOUT)
            (GO ILOOP))
      (SETO EXPERIMENT-LIST (CDR EXPERIMENT-LIST))
      (COND
        ((NULL EXPERIMENT-LIST)
          NIL)
        (T (GO OLOOP))
    )
(PRINT-RESULTS
  (LAMBDA (M-LIST INSYM OUTSYM)
    (PROGN (TERPRI)
      (PRIN1 "The machines conjectured from the observed input")
      (PRIN1 INSYM)
      (PRIN1 "and observed output")
      (PRIN1 OUTSYM)
      (PRIN1 "are!")
      (TERPRI)
      (TERPRI)
      (PRINT-MACHINES M-LIST)
      (TERPRI))
    (PRINT-RESULTS)
    -)

(PRINT-MACHINES
  (LAMBDA (M-LIST)
    (PROGN (PRINT-MACHINE-CHANGE (CAR M-LIST))
      (COND
        ((NULL (CDR M-LIST))
         NIL)
        (T (PRINT-MACHINES (CDR M-LIST)))
      (PRINT-MACHINES)
      -)
(PRINT-MACHINE-CHANGE)

(PRINT-MACHINE-CHANGE
 [LAMBDA (MACHINE)]

(PRIN1 "Machine")
(PRIN1 (PACK (CDR (UNPACK MACHINE))
(PRIN1 "is conjectured from Machine")
(PRIN1 (PACK (CDR (UNPACK (GETPROP MACHINE (QUOTE PARENT-MACHINE))
(PRIN1 "with a transition from")
(PRIN1 (GETPROP MACHINE (QUOTE PREVIOUS-STATE)))
(PRIN1 "to")
(PRIN1 (GET-CURRENT-STATE MACHINE))
(TERPRI))

(PRINT-MACHINE-CHANGE)
(INITIAL-MACHINE
  (LAMBDA NIL
    (PROG (MACHINE S-NAME)
      (SETQ MACHINE (MACHINE-NAME NIL))
      (PUT-STATE-COUNT MACHINE (QUOTE 1))
      (PUT-CURRENT-STATE MACHINE (QUOTE 1))
      (SETQ S-NAME (STATE-NAME MACHINE (QUOTE 1)))
      (PUT-STATE-LIST MACHINE (LIST S-NAME))
      (PUT-TRANSITION-LIST S-NAME NIL)
      (RETURN MACHINE))
    (INITIAL-MACHINE)
    -)

(MACHINE-NAME
  (LAMBDA NIL
    (GENSYM (QUOTE M3))
    (MACHINE-NAME)
    -)

-
```lisp
(GET-ALL-TRANSITIONS)

(APPLY (QUOTE APPEND)
  (MAPCAR T-LIST (QUOTE (LAMBDA (TLE)
    (COND
      ((EQUAL INSYM (CAR TLE))
        (LIST TLE))
    (GET-ALL-TRANSITIONS)
  )))

(GET-CURRENT-STATE)

(GETPROP MACHINE (QUOTE CURRENT-STATE))
(GET-CURRENT-STATE)
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(GET-OBSERVED-IO-PAIR
  #'(LAMBDA NIL
     (PROG (PAIR)
       (SETQ PAIR (CAR OBSERVED-IO-BEHAVIOR))
       (SETQ OBSERVED-IO-BEHAVIOR (CDR OBSERVED-IO-BEHAVIOR))
       (RETURN PAIR)))

(GET-OBSERVED-IO-PAIR)

(GET-STATE-COUNT
  #'(LAMBDA (MACHINE)
     (GETPROP MACHINE (QUOTE STATE-COUNT))
     (GET-STATE-COUNT))

(GET-STATE-LIST
  #'(LAMBDA (MACHINE)
     (GETPROP MACHINE (QUOTE STATE-LIST))
     (GET-STATE-LIST))

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(GET-SYMBOL-COUNT)

(GET-SYMBOL-COUNT
  (LAMBDA (SN SYMBOL))

  (PROG (SL)
    (SETQ SL (SASSOC SYMBOL (GET-SYMBOL-LIST SN)))
    (COND
      ((NULL SL)
       (RETURN NIL))
      (T (RETURN (CADR SL)))
  (GET-SYMBOL-COUNT)

-PP*(GET-SYMBOL-LIST)

(GET-SYMBOL-LIST
  (LAMBDA (SN)
    (GETPROP SN (QUOTE SYMBOL-COUNT))
  (GET-SYMBOL-LIST)

-PP*(GET-TRANSITION-LIST)

(GET-TRANSITION-LIST
  (LAMBDA (STATE-ATOM)
    (GETPROP STATE-ATOM (QUOTE TRANSITIONS))
  (GET-TRANSITION-LIST)
(PUT-CURRENT-STATE)

(PUT-CURRENT-STATE
 \[\text{CLAMBDA (MACHINE STATE)}\]

(PUTPROP MACHINE (QUOTE CURRENT-STATE)
 STATE])
(PUT-CURRENT-STATE)
-

_PP#(PUT-STATE-COUNT)

(PUT-STATE-COUNT
 \[\text{CLAMBDA (MACHINE COUNT)}\]

(PUTPROP MACHINE (QUOTE STATE-COUNT)
 COUNT])
(PUT-STATE-COUNT)
-

_PP#(PUT-STATE-LIST)

(PUT-STATE-LIST
 \[\text{CLAMBDA (MACHINE S-LIST)}\]

(PUTPROP MACHINE (QUOTE STATE-LIST)
 S-LIST])
(PUT-STATE-LIST)
-
(_PP*(PUT-SYMBOL-COUNT))

(PUT-SYMBOL-COUNT
  CLAMBDA (SN SYMBOL COUNT))

(PROG (OSL OSC)
  (GET-SYMBOL-LIST SN))

(SETQ OSC (GET-SYMBOL-COUNT SN SYMBOL))

(COND
  [(NULL OSC)
   (COND
    [(NULL OSL)
     (PUT-SYMBOL-LIST SN (LIST (LIST SYMBOL COUNT))]
    [T (PUT-SYMBOL-LIST SN (APPEND OSL (LIST (LIST SYMBOL COUNT))]
     (T (PUT-SYMBOL-LIST SN (SUBST (LIST SYMBOL COUNT)
      (LIST SYMBOL OSC) OSL))]
    [T (PUT-SYMBOL-LIST SN (SUBST (LIST SYMBOL COUNT)
      (LIST SYMBOL OSC) OSL))]
    )]
  )]

(PUT-SYMBOL-COUNT)

(_PP*(PUT-SYMBOL-LIST))

(PUT-SYMBOL-LIST
  CLAMBDA (SN SL)
  (PUTPROP SN (QUOTE SYMBOL-COUNT)
    SL))

(PUT-SYMBOL-LIST)

(_PP*(PUT-TRANSITION-LIST))

(PUT-TRANSITION-LIST
  CLAMBDA (STATE-ATOM T-LIST)
  (PUTPROP STATE-ATOM (QUOTE TRANSITIONS)
    T-LIST))

(PUT-TRANSITION-LIST)
(NEW-MACHINE-LIST
  (LAMBDA (M-LIST INSYM OUTSYM)
    (APPEND (CREATE-DESCendant-MACHINES (CAR M-LIST)
           INSYM OUTSYM)
    (COND
      ((NULL (CDR M-LIST))
       NIL)
      (T (NEW-MACHINE-LIST (CDR M-LIST)
           INSYM OUTSYM))
    (NEW-MACHINE-LIST)
  )
)

# edited:
*12-NOV-82 23:01*
(_PP* (CREATE-DESCENDANT-MACHINES))

(CREATE-DESCENDANT-MACHINES
  Lambda (MACHINE INSYM OUTSYM))

(APPEND (RULE1-MACHINES MACHINE INSYM OUTSYM)
  (COND
    ((NEW-STATE-OK? MACHINE INSYM OUTSYM)
     (LIST (RULE2-MACHINE MACHINE INSYM OUTSYM)))
  (CREATE-DESCENDANT-MACHINES))

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(RULE1-MACHINES)

(RULE1-MACHINES)

(LAMBDA (MACHINE INSYM OUTSYM)

(PROG (STATE NEW-MACHINE NEW-LIST)

  (for STATE from 1 to (GET-STATE-COUNT MACHINE)
   do (COND
       ((NOT (TRANSITION-CONFLICT? MACHINE STATE INSYM OUTSYM))
        (SETQ NEW-MACHINE (CREATE-MACHINE MACHINE))
        (NEW-TRANSITION NEW-MACHINE STATE INSYM OUTSYM)
        (PUT-CURRENT-STATE NEW-MACHINE STATE)
        (SETQ NEW-LIST (APPEND NEW-LIST (LIST NEW-MACHINE))
        (RETURN NEW-LIST))
    )

(RETURN NEW-LIST))

(RULE1-MACHINES)
(RULE2-MACHINE
  [LAMBDA (MACHINE INSYM OUTSYM)]

  (FROG (NEW-MACHINE STATE NSN))
  (SETO NEW-MACHINE (CREATE-MACHINE MACHINE))
  (SETO STATE (ADDI (GET-STATE-COUNT MACHINE)))
  (SETO NSN (STATE-NAME NEW-MACHINE STATE))
  (NEW-TRANSITION NEW-MACHINE STATE INSYM OUTSYM)
  (PUT-STATE-LIST NEW-MACHINE (APPEND (GET-STATE-LIST NEW-MACHINE)
    (LIST NSN)))
  (PUT-STATE-COUNT NEW-MACHINE STATE)
  (PUT-CURRENT-STATE NEW-MACHINE STATE)
  (RETURN NEW-MACHINE))

  (RULE2-MACHINE)
\[ \text{PP* (NEW-STATE-OK?)} \]

\( \text{(NEW-STATE-OK?} \)
\( \text{IMBDA (MACHINE INSYM OUTSYM)} \)
\( \text{(* edited: } \)
\( \text{*28-NOV-82 16:36')} \)
\( \text{(PROG (S-CRITERIA TRANS)} \)
\( \text{(SETO S-CRITERIA (IBGREATERF MAXIMUM-STATES (GET-STATE-COUNT MACHINE)))} \)
\( \text{(SETO TRANS (GET-ALL-TRANSITIONS (GET-TRANSITION-LIST}} \)
\( \text{(STATE-NAME MACHINE (GET-CURRENT-STATE} \)
\( \text{MACHINE)))} \)
\( \text{INSYM))} \)
\( \text{(COND} \)
\( \text{((EQUAL S-CRITERIA NIL)} \)
\( \text{(RETURN NIL))} \)
\( \text{((NULL TRANS)} \)
\( \text{(RETURN T))} \)
\( \text{((OUTPUT-THERE? TRANS OUTSYM)} \)
\( \text{(RETURN NIL))} \)
\( \text{(T (RETURN T))} \)
\( \text{(NEW-STATE-OK?)} \)
\( \text{-} \)
(TRANSITION-CONFLICT?
  (LAMBDA (MACHINE STATE INSYM OUTSYM))
  (# edited:
 '29-NOV-82 20:25')

(PROG (TRANS)
  (SETO TRANS (GET-ALL-TRANSITIONS (GET-TRANSITION-LIST
    (STATE-NAME MACHINE
      (GET-CURRENT-STATE
        MACHINE)))
      INSYM))

(COND
  ((NULL TRANS)
    (RETURN NIL))
  ((OUTPUT-THERE? TRANS OUTSYM)
    (COND
     (EQUAL STATE (APPLY (QUOTE OR)
       (MAPCAR TRANS
         (QUOTE (LAMBDA
           (X)
           (COND
             ((EQUAL OUTSYM
               (CADDR X))
             (RETURN T))
             (RETURN NIL))))
       (STATE-THERE? TRANS STATE)
       (RETURN T))
     (T (RETURN NIL))))

  (TRANSITION-CONFLICT?)

  )))
_PP*(OUTPUT-THERE?)

(OUTPUT-THERE?
  _LAMBDA (T-LIST OUTSYM)

(APPLY (QUOTE OR)
  (MAPCAR T-LIST (QUOTE LAMBDA (TLE))
   (EQUAL OUTSYM (CADDR TLE))

(OUTPUT-THERE?)

_PP*(STATE-THERE?)

(STATE-THERE?
  _LAMBDA (T-LIST STATE)

(APPLY (QUOTE OR)
  (MAPCAR T-LIST (QUOTE LAMBDA (TLE))
   (EQUAL STATE (CADDR TLE))

(STATE-THERE?)

-
_PP*(CREATE-MACHINE)

(CREATE-MACHINE
 CLAMBDA (BASE-MACHINE)

 (PROG (NEXT-MACHINE)
   (SETQ NEXT-MACHINE (MACHINE-NAME NIL))
   (PUT-STATE-COUNT NEXT-MACHINE (GET-STATE-COUNT BASE-MACHINE))
   (PUT-CURRENT-STATE NEXT-MACHINE (GET-CURRENT-STATE BASE-MACHINE))
   (PUT-STATE-LIST NEXT-MACHINE (GET-STATE-LIST BASE-MACHINE))
   (TRANSFER-STATE-DEFINITIONS BASE-MACHINE NEXT-MACHINE)
   (PUTPROP NEXT-MACHINE (QUOTE PREVIOUS-STATE)
       (GET-CURRENT-STATE NEXT-MACHINE))
   (PUTPROP NEXT-MACHINE (QUOTE PARENT-MACHINE)
       BASE-MACHINE)
   (RETURN NEXT-MACHINE)
)

(CREATE-MACHINE)
(TRANSFER-STATE-DEFINITIONS)

(LAMBDA (OLD-MACHINE NEW-MACHINE)  
  (PROG (OLD-S-LIST NEW-S-LIST STATE OSN NSN)  
    (setq OLD-S-LIST (GET-STATE-LIST OLD-MACHINE)) 
    (setq NEW-S-LIST OLD-S-LIST) 
    (for STATE from 1 to (GET-STATE-COUNT OLD-MACHINE)  
      do ((setq OSN (STATE-NAME OLD-MACHINE STATE))  
        (cond  
          ((not (null OSN))  
            (setq NSN (STATE-NAME NEW-MACHINE STATE))  
            (setq NEW-S-LIST (SUBST NSN OSN NEW-S-LIST))  
            (put-TRANSITION-LIST NSN (GET-TRANSITION-LIST OSN))  
            (put-SYMBOL-LIST NSN (GET-SYMBOL-LIST OSN))  
            (put-STATE-LIST NEW-MACHINE NEW-S-LIST))  
        )))))

(TRANSFER-STATE-DEFINITIONS)

(STATE-NAME)

(LAMBDA (MACHINE STATE)  
  (PACK (APPEND (UNPACK MACHINE)  
    (UNPACK (QUOTE -S))  
    (UNPACK STATE))  
  )

(STATE-NAME)
(NEW-TRANSITION
  (CLARINDA (MACHINE STATE INSYM OUTSYM))
  (# edited: 28-NOV-82 16:19*)
  (PROG (N-TRAN S-NAME)
    (SETQ N-TRAN (LIST INSYM STATE OUTSYM))
    (SETQ S-NAME (STATE-NAME MACHINE (GET-CURRENT-STATE MACHINE)))
    (UPDATE-TRANSITIONS MACHINE STATE INSYM OUTSYM)
    (PUTPROP MACHINE (QUOTE LAST-TRANSITION)
      N-TRAN)
    (RETURN N-TRAN))
  (NEW-TRANSITION)
  -
(UPDATE-TRANSITIONS)

* edited: 29-N0V-82 09:25"

(PRQ (S-NAME T-LIST TRANS R-TRAN)

(SETQ S-NAME (STATE-NAME MACHINE (GET-CURRENT-STATE MACHINE)))

(SETQ T-LIST (GET-TRANSITION-LIST S-NAME))

(SETQ TRANS (GET-ALL-TRANSITIONS T-LIST INSYM))

(COND

((NULL TRANS)

(PUT-TRANSITION-LIST S-NAME

(APPEND T-LIST

(LIST (LIST INSYM STATE OUTSYM 1)

(PUT-SYMBOL-COUNT S-NAME INSYM 1))

(T (SETQ R-TRAN

(APPLY (QUOTE APPEND)

(MAPCAR TRANS (QUOTE (LAMBDA

(X)

(COND

((AND (EQUAL STATE

(CADR X))

(EQUAL OUTSYM

(CADDR X)))

X))

(COND

((NULL R-TRAN)

(PUT-TRANSITION-LIST S-NAME

(APPEND T-LIST

(LIST (LIST INSYM STATE OUTSYM 1)

(PUT-SYMBOL-COUNT S-NAME INSYM (ADD1 (GET-SYMBOL-COUNT S-NAME INSYM))

(T (PUT-TRANSITION-LIST S-NAME

(SUBST (LIST INSYM STATE OUTSYM

(ADD1 (CADDR R-TRAN)))

R-TRAN

(GET-TRANSITION-LIST S-NAME)))

(PUT-SYMBOL-COUNT S-NAME INSYM (ADD1 (GET-SYMBOL-COUNT S-NAME INSYM))))

(UPDATE-TRANSITIONS)
Enter an i-o experiment, NIL when done! ((A 0)(A 1)(A 0)(A 1)(A 1)(A 0)(A 0) (A 0)(A 0))
Enter an i-o experiment, NIL when done: ()


The machines conjectured from the observed input A and observed output 0 are:

Machine 6 is conjectured from Machine 5 with a transition from 1 to 1
Machine 7 is conjectured from Machine 5 with a transition from 1 to 2

The machines conjectured from the observed input A and observed output 1 are:

Machine 8 is conjectured from Machine 6 with a transition from 1 to 2
Machine 9 is conjectured from Machine 7 with a transition from 2 to 1
Machine 10 is conjectured from Machine 7 with a transition from 2 to 2

The machines conjectured from the observed input A and observed output 1 are:

Machine 11 is conjectured from Machine 8 with a transition from 2 to 1
Machine 12 is conjectured from Machine 8 with a transition from 2 to 2
Machine 13 is conjectured from Machine 9 with a transition from 1 to 1
Machine 14 is conjectured from Machine 10 with a transition from 2 to 2

The machines conjectured from the observed input A and observed output 1 are:

Machine 15 is conjectured from Machine 11 with a transition from 1 to 2
Machine 16 is conjectured from Machine 12 with a transition from 2 to 2
Machine 17 is conjectured from Machine 13 with a transition from 1 to 1
Machine 18 is conjectured from Machine 14 with a transition from 2 to 2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The machines conjectured from the observed input A and observed output 0 are:

Machine 19 is conjectured from Machine 15 with a transition from 2 to 2
Machine 20 is conjectured from Machine 16 with a transition from 2 to 1
Machine 21 is conjectured from Machine 17 with a transition from 1 to 2
Machine 22 is conjectured from Machine 18 with a transition from 2 to 1

The machines conjectured from the observed input A and observed output 0 are:

Machine 23 is conjectured from Machine 19 with a transition from 2 to 2
Machine 24 is conjectured from Machine 20 with a transition from 1 to 1
Machine 25 is conjectured from Machine 21 with a transition from 2 to 2
Machine 26 is conjectured from Machine 22 with a transition from 1 to 2

The machines conjectured from the observed input A and observed output 1 are:

Machine 27 is conjectured from Machine 23 with a transition from 2 to 1
Machine 28 is conjectured from Machine 24 with a transition from 1 to 2
Machine 29 is conjectured from Machine 25 with a transition from 2 to 1
Machine 30 is conjectured from Machine 26 with a transition from 2 to 2

The machines conjectured from the observed input A and observed output 1 are:

Machine 31 is conjectured from Machine 27 with a transition from 1 to 2
Machine 32 is conjectured from Machine 28 with a transition from 2 to 2
Machine 33 is conjectured from Machine 29 with a transition from 1 to 1
Machine 34 is conjectured from Machine 30 with a transition from 2 to 2

The machines conjectured from the observed input A and observed output 1 are:

Machine 35 is conjectured from Machine 31 with a transition from 2 to 1
Machine 36 is conjectured from Machine 32 with a transition from 2 to 2
Machine 37 is conjectured from Machine 33 with a transition from 1 to 1
Machine 38 is conjectured from Machine 34 with a transition from 2 to 2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The machines conjectured from the observed input A and observed output 0 are:

Machine 39 is conjectured from Machine 35 with a transition from 1 to 1
Machine 40 is conjectured from Machine 36 with a transition from 2 to 1
Machine 41 is conjectured from Machine 37 with a transition from 1 to 2
Machine 42 is conjectured from Machine 38 with a transition from 2 to 1

EXPERIMENT 2: \((A_0 \ A_1 \ A_0 \ A_1 \ A_1 \ A_0 \ A_0)\)

The machines conjectured from the observed input A and observed output 0 are:

Machine 43 is conjectured from Machine 39 with a transition from 1 to 1
Machine 44 is conjectured from Machine 40 with a transition from 1 to 1
Machine 45 is conjectured from Machine 41 with a transition from 1 to 2
Machine 46 is conjectured from Machine 42 with a transition from 1 to 2

The machines conjectured from the observed input A and observed output 1 are:

Machine 47 is conjectured from Machine 43 with a transition from 1 to 2
Machine 48 is conjectured from Machine 44 with a transition from 1 to 2
Machine 49 is conjectured from Machine 45 with a transition from 2 to 1
Machine 50 is conjectured from Machine 46 with a transition from 2 to 2

The machines conjectured from the observed input A and observed output 0 are:

Machine 51 is conjectured from Machine 47 with a transition from 2 to 2
Machine 52 is conjectured from Machine 48 with a transition from 2 to 1
Machine 53 is conjectured from Machine 49 with a transition from 1 to 2
Machine 54 is conjectured from Machine 50 with a transition from 2 to 1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The machines conjectured from the observed input $A$ and observed output $1$ are:

Machine 55 is conjectured from Machine 51 with a transition from 2 to 1
Machine 56 is conjectured from Machine 52 with a transition from 1 to 2
Machine 57 is conjectured from Machine 53 with a transition from 2 to 1
Machine 58 is conjectured from Machine 54 with a transition from 1 to 1

The machines conjectured from the observed input $A$ and observed output $1$ are:

Machine 59 is conjectured from Machine 55 with a transition from 1 to 2
Machine 60 is conjectured from Machine 56 with a transition from 2 to 2
Machine 61 is conjectured from Machine 57 with a transition from 1 to 1
Machine 62 is conjectured from Machine 58 with a transition from 1 to 1

The machines conjectured from the observed input $A$ and observed output $1$ are:

Machine 63 is conjectured from Machine 59 with a transition from 2 to 1
Machine 64 is conjectured from Machine 60 with a transition from 2 to 2
Machine 65 is conjectured from Machine 61 with a transition from 1 to 1
Machine 66 is conjectured from Machine 62 with a transition from 1 to 1

The machines conjectured from the observed input $A$ and observed output $0$ are:

Machine 67 is conjectured from Machine 63 with a transition from 1 to 1
Machine 68 is conjectured from Machine 64 with a transition from 2 to 1
Machine 69 is conjectured from Machine 65 with a transition from 1 to 2
Machine 70 is conjectured from Machine 66 with a transition from 1 to 2

The machines conjectured from the observed input $A$ and observed output $0$ are:

Machine 71 is conjectured from Machine 67 with a transition from 1 to 1
Machine 72 is conjectured from Machine 68 with a transition from 1 to 1
Machine 73 is conjectured from Machine 69 with a transition from 2 to 2
Machine 74 is conjectured from Machine 70 with a transition from 2 to 1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The machines conjectured from the observed input A and observed output 0 are:

- Machine 75 is conjectured from Machine 71 with a transition from 1 to 1
- Machine 76 is conjectured from Machine 72 with a transition from 1 to 1
- Machine 77 is conjectured from Machine 73 with a transition from 2 to 2
- Machine 78 is conjectured from Machine 74 with a transition from 1 to 2

- Machine 79 is conjectured from Machine 75 with a transition from 1 to 1
- Machine 80 is conjectured from Machine 76 with a transition from 1 to 1
- Machine 81 is conjectured from Machine 77 with a transition from 2 to 2
- Machine 82 is conjectured from Machine 78 with a transition from 2 to 1

NIL
_GET-STATE-LIST(M0081)
(M0081-S1 M0081-S2)
-

_GET-TRANSITION-LIST(M0081-S1)
((A 2 0 6) (A 1 1 6))
_GET-SYMBOL-LIST(M0081-S1)
((A 12))
-

_GET-TRANSITION-LIST(M0081-S2)
((A 1 1 4) (A 2 0 4))
_GET-SYMBOL-LIST(M0081-S2)
((A 8))
-
-

_GET-STATE-LIST(M0082)
(M0082-S1 M0082-S2)
-

_GET-TRANSITION-LIST(M0082-S1)
((A 2 0 5) (A 1 1 3))
_GET-SYMBOL-LIST(M0082-S1)
((A 8))
-

_GET-TRANSITION-LIST(M0082-S2)
((A 2 1 7) (A 1 0 5))
_GET-SYMBOL-LIST(M0082-S2)
((A 12))
-
BIBLIOGRAPHY


