INFORMATION AND CONTROL 2, 364–379 (1959)

Two Notes on Machine "Learning"

HENRIK H. MARTENS

Bell Telephone Laboratories, Inc., Murray Hill, New Jersey

The plan of a program that enables a computer to "learn" to play tic-tac-toe, and related 3 × 3 board games, is described. The programmed computer has no built-in knowledge of the game to be played, except for a rule for determining legal moves. It specifically does not "know" what constitutes a win, loss, or draw, but must be informed of the outcome at the end of each play. Experience indicates that a fair competence in tic-tac-toe playing is reached after 30 to 50 plays.

Generalizing from this example of a "learning machine," the notion of an L-automaton is introduced via a formal, behaviouristic definition, in an attempt to give an abstract characterization of machine "learning." A solution to the design problem for a general class of L-automata is presented.

INTRODUCTION

The possibility of realizing learning or adaptive behavior in automata has, in recent years, attracted considerable attention, and has provided food for much speculation under the general heading "Can Machines Think?". In the following pages we shall attempt to prepare the way for a theory of a class of automata whose behavior is akin to learning. The definition of this class is behavioristic: it does not prescribe the internal structure of the automata, and may thus also be taken as an attempted characterization of "learning" appropriate for the application to automata. Our main conclusion is that the design problem for this class of automata is, in a certain sense, trivial.

It should be emphasized that our object is to study a form of machine behavior believed to be of interest in its own right; we make no attempt to set up models for human or animal brain functions. Hence the word "learning" must be understood merely as a suggestive designation of the kind of behavior we have in mind, and which remains to be characterized more precisely. As a concession to the champions of the supremacy of mind over matter we have adopted the designations "L-automaton" and "L-behavior" in our formal work.

Because of the inherent abstraction of the definition of L-automata, it appeared desirable to introduce and motivate the discussion by a note describing a game-"learning" program recently written for the Bell Telephone Laboratories' Leprechaun computer. The program was designed to enable the computer to "learn" to play any of a class of games playable on a 3 × 3 board.

Note I: A Computer Program for Game-"Learning"

1. DESCRIPTION OF PROGRAM

1.1 Introductory Remarks

As we hope to show in our second note, it is possible to give a general characterization of machine "learning," and to give a general solution to the design problem for "learning machines," provided certain finiteness requirements are satisfied.

It is hardly to be expected that such a general solution will be of much practical significance. The interesting cases appear only when the "learning" task is sufficiently well defined so that the inputs and outputs of the machine may be assumed to have a definite structure. In the game-"learning" program, to be described presently, it is assumed that the input and output symbols are related to the positions of marks on a 3 × 3 board, which may be thought of as the familiar tic-tac-toe board. Under this assumption it is possible to program the machine to take advantage, wherever possible, of structure properties such as symmetry, reversibility of positions, etc.

We should like, at this point, to draw attention to the fact that the game-"learning" program satisfies one (but hardly more than this) criterion for a useful machine: it is designed to operate in real time, i.e. to play against a human opponent, and acquires a reasonable skill in the course of 30 to 50 plays. While we have no illusions about the significance of this fact in the present context, it seems to us that this criterion is one to be kept in mind when evaluating the "learning efficiency" of a proposed machine. One should examine the time required to "learn" a task in a natural environment rather than in a simulated environment where the time scale may be completely distorted.

1.2 Heuristic Description of Program

The basic frame of reference is the 3 $\times$ 3 board. We assume, furthermore, that the games to be played are subject to the further restriction that

(A) The contestants play by alternately marking one, and only one, previously unmarked field.

(B) The evaluation of the game is a fixed function of the end configuration, the final move having been made by the winner (except in case of a draw).

One may note, in particular, that no assumption has been made about any symmetry of the game.

The programmed computer is initially "ignorant" of the rules of the game, except as specified under (A) and (B) above. Thus, it can determine legal moves, but must "learn" the winning combinations by experience. To do so, it selects its moves guided by a few heuristic ideas. These are formulated in six rules with an order of preference.

The *first* rule calls for a perfect repetition of a previous win, if possible. That is: if by marking a certain field the machine produces a perfect match to a configuration listed as a win in its memory, it makes that mark.

The *second* rule calls for a partial match of a previous win. In this case the marks made by the machine in a previous win are assumed to be a subset of those made by the machine in the present play plus some mark which the machine then selects as its next move.

The *third* rule reflects an attempt to imitate the opponent by applying a procedure analogous to that in the second rule to the opponent's marks in a previously lost play.

The *fourth* rule calls for a blocking move to prevent the opponent from repeating a previous disaster. In this case the opponent's marks are assumed to be such that the addition of another mark will produce a set containing the opponent's marks in a previously lost play, and the machine selects the move that will prevent the opponent from completing the inclusion.

The fifth and sixth rules refer to situations where none of the previous rules apply. In order to describe these rules we must first introduce the notion of an *admissible* move. When the machine is confronted with a board configuration and asked to make a move, it first determines what the possible legal moves are. It then scans its memory to determine whether any of the legal moves would result in a loss. The end configuration of any lost play is recorded in memory as a hopeless configuration. If, by making one of the legal moves, the machine would leave an opportunity for the opponent to make a move and match a hopeless configuration, that legal move is recorded as inadmissible. If it turns out that all legal moves are inadmissible, the board configuration is entered in memory as a hopeless one.

Rules *five* and *six* then call for the selection of an admissible move or a legal move, respectively.

To make things specific, arbitrary rules are made for selecting moves, where several moves could be made under the same rule. Thus the fields of the board are numbered, and the lowest numbered field is always selected. If a rule applies to several memory entries, it is applied to the last entry. Finally, as mentioned above, the rules are given in order of preference.

We have so far said nothing about symmetries. If we consider the board as a square divided into nine fields, as for tic-tac-toe, there are eight obvious geometric symmetry transformations. In matching configurations, the machine will at first make no distinction between symmetric configurations. However, if a perfect match is made according to the first rule, and the expected win is not forthcoming, the machine will record the symmetry transformation as invalid and make no further use of it.

It should be noted that although the rules are reasonable for tic-tac-toe, games may be imagined where some of the rules would be of no help. This is only to be expected. One may think of the machine as having a tendency to make rash, and rather simple-minded, assumptions about the game and to act accordingly. The justification for this is simply that the alternative to such action is an arbitrary or random selection of moves, and that a selection based on a hasty assumption is likely to be no worse than a random selection if it does not work. However, it *may* work.

The choice of a simple 3 × 3 board was dictated by the size of the computer at our disposal. An awkward consequence of this choice was the difficulty of finding suitable games other than tic-tac-toe. The only other game tried on the machine was a rather trivial version of Hex. An interesting discussion of this game will be found in the July 1957 issue of the "Scientific American." The game may be played on boards of varying sizes, the 3 × 3 board being made up of nine hexagons as shown in the figure. The object of the game is for contestant A to place marks

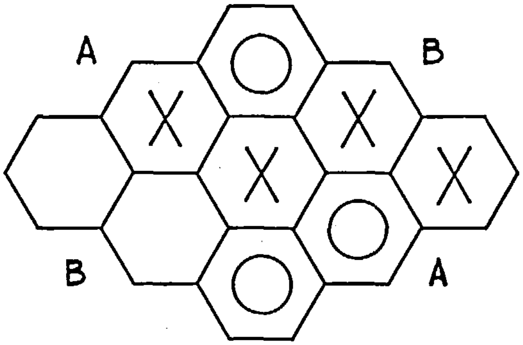

Fig. 1. 3 × 3 Hex Board. A win for A (crosses) is indicated

in adjacent hexagons forming a connected line between the sides marked A. Contestant B should attempt the same for the sides marked B.

It will be noted that if the board is deformed into a square, only one of the symmetries of the latter (other than the identity) is valid for the game, and reversion of positions is meaningless. Hence this provides an example of a game that is quite incompatible (in the sense of our second note) with tic-tac-toe. It should be clear, however, that although Hex is played on hexagons and tic-tac-toe on squares, it is a matter of indifference to the machine what the form of the nine fields are, and the Hex game obviously satisfies conditions (A) and (B).

1.3 Formal Description

In order to state our rules more precisely it is convenient to introduce the following notation:

The machine's *position* on the board is given by specifying the fields it has marked, and is denoted by an "X." The opponent's position is denoted by an "O." A *configuration*, C, is given by a pair (X,O).

The nine fields of the board are numbered. A machine's move into the ith field is denoted by "M_i." An opponent's move into the same field is denoted by "M_i'."

A *transformation*, T, is a symmetry transformation of the board, considered as a square. During the course of, or prior to, a play certain transformations may be designated as *invalid*.

Configurations and positions may be operated on in three ways.

(a) Application of a move:

$$C \rightarrow M_iC \quad \text{or} \quad (X,O) \rightarrow (M_iX,O).$$

$$C \rightarrow M_i'C \quad \text{or} \quad (X,O) \rightarrow (X,M_i'O).$$

(b) Application of a transformation:

$$C \rightarrow TC \quad \text{or} \quad (X,O) \rightarrow (TX,TO).$$

(c) Inversion of positions:

$$C \rightarrow C' \quad \text{or} \quad (X,O) \rightarrow (O',X').$$

The inversion operation consists in changing a contestant's position into his opponent's position by "reversing" the mark of every marked field.

The relation of *inclusion* between configurations is taken as self-explanatory:

$$C_1 \subseteq C_2 \quad \text{or,} \qquad X_1 \subseteq X_2 \quad \text{and} \quad O_1 \subseteq O_2$$

Given a configuration, C, a move, M_i, is said to be *clear* (in C) if the ith field is unmarked.

Configurations are listed in memory as *won* or *hopeless*. Given a configuration, C, a clear move, M_i, is said to be *inadmissible* (in C) if there exist a clear move, M_j, a hopeless configuration, C_h, and a valid transformation, T, such that

$$M_j'M_iC = TC_h$$

Otherwise M_i is said to be *admissible*.

During the course of a play, and before a decision is reached, the inputs to the machine specify the move selected by the opponent. This move is added to the machine's record of the current play to form the *present configuration*, $C = (X,O)$.

The machine's move is selected by means of the following six rules:

Let C_h and C_w be hopeless and won configurations listed in the memory. Let T be a valid transformation. Let M_i be a clear move (rules 1 and 6), or an admissible move, (rules 2 through 5).

Move M_i is selected by the rule if

1. $M_iC = TC$, or
2. $TX_w \subseteq M_iX$, or
3. $TO_h' \subseteq M_iX$, or
4. $TO_h \subseteq M_i'O$, or

5. M_i is the lowest-numbered admissible move, or
6. M_i is the lowest-numbered clear move.

These rules must be supplemented by the following conditions.

(a) Rules 1 through 5 are applied only if admissible moves exist.

(b) The move selected by a rule shall be that determined by the last memory entry to which the rule applies. The move selected by the machine shall be that selected by the lowest-numbered rule.

(c) Rules 5 and 6 are subjected to the following condition: Whenever a move is made under one of these rules, it is recorded in a register. On the next application of the rule the lowest-numbered move not in the register shall be selected. If no such move exists the register is erased and the process repeated.

1.4 Memory Record Rules

At the end of a play the input to the machine specifies the opponent's final move, if any, and designates the play as a win, loss or draw for the machine.

When presented with this information the machine stores the present configuration of a win as *won*, unless the last machine move was made under rule 1. It stores the present configuration of a lost play as *hopeless*, unless the last machine move was made under rule 6. If the play is a draw, or if the exceptions noted above are satisfied, the machine stores nothing.

If, in the course of a play, the present configuration has no admissible move, that configuration is stored as hopeless.

If rule 1 has been applied and the next input does not designate the play as a win, the transformation used in the application of the rule is recorded as invalid. If the identity transformation is recorded as invalid a special output occurs, and the machine prepares to erase its memory.

2. PERFORMANCE

Although our experience in playing against the machine is somewhat limited, sufficient evidence is available to permit a fair estimate of its capabilities and weaknesses.

2.1 Influence of Opponent

It should be emphasized that the "knowledge" acquired by the machine about the game is entirely empirical. Hence, the ability of the machine to cope with a situation depends largely on the extent to which it has previously been exposed to similar situations. In the case of tic-tac-toe, for instance, the machine will deliberately block an opponent's win along a diagonal only if it has a record of a previous disaster of this kind. As a result, it is possible to keep the machine in ignorance about certain winning combinations indefinitely. In other words: given an integer, N, no matter how large, it is always possible to play N plays against the machine and still be able to win. In order to do so, however, the opponent must either lose or draw most of these plays for large N.

The important thing, of course, is that there exists an upper limit on the number of losses that the machine will take, and experience to date indicates that this number, for tic-tac-toe, is of the order of 50 to 60. It may be possible to increase this number by an elaborate scheme of playing, but the problem of doing so is rather complicated.

It follows from these remarks that the machine's ability to play is closely related to the opponent's ability to present it with a tough opposition. Starting from scratch, the machine will only develop enough skill to beat its opponent, and no matter how long it is exposed to a poor opponent, a better opponent may still have a chance to beat it for a short while. Eventually, however, even the expert must be satisfied with a draw.

2.2 Peculiarities

One of the interesting aspects of the machine is that its behavior during the "learning process" is often quite different from that of a human player. The process of working backwards from the end configuration results in a rather weak opening strategy in early plays, and where the key moves are the early moves, the machine takes longer to "learn." Since it insists on trying all possible countermoves in the later stage of a play before it modifies its early moves, the machine will sometimes appear to have forgotten previous losses.

Moreover, since the machine does not have any prejudices about the winning combinations, it sometimes appears to neglect to block an "obvious" opportunity for the opponent to win. Thus having lost to a diagonal three-on-a-row with an extra mark on the board, it will not block a pure diagonal three-on-a-row. This sometimes puzzles observers who do not realize that, from the machine's point of view, the extra mark may be just as responsible for the disaster as the three diagonal marks. It is possible to utilize this in delaying the "learning process" by playing so as to win with more than three marks early in the game.

TABLE I

	Plays	Wins	Losses	Draws
1. LLLLLLLLLLLLLLLLLLLL	20	0	20	0
LLDDDLLLLDDDDDDDDLLL	20	0	9	11
LLDLDLDDLLLLDDDDDDDL	20	0	9	11
DLLLLLDDDDDWLDDLLDD	19	1	8	10
	—	—	—	—
	79	1	46	32
2. LDLLDDLDDLLLLLLLWWDW	20	3	11	6
LDDLLLLLLWLLWDLWLLWW	20	5	12	3
DLLLLLWDDL	10	1	6	3
	—	—	—	—
	50	9	29	12

As a result, the best strategy against the machine may be quite different from the best strategy against a human player. Thus, in tic-tac-toe, the best opening move against the machine is a corner move, or a move into the middle of a side, since these destroy the board symmetry and result in the greater number of distinct opening countermoves. Typical examples of what can be accomplished in this manner are shown in Table I.

This table represents the results of two series of tic-tac-toe plays starting from scratch. The first string was obtained by letting the opponent have the first move. The second was obtained by giving the machine the first move. The difference between the two strings illustrate quite effectively the degree of control the first player has over the play. In both of these examples the playing was continued until it was felt that all possible ways of winning had been exhausted. At the end of the first string a few plays were played, without erasing the memory, giving the machine the first move. The results were as follows:

LLLLDDDDDWL.

The interesting aspect of this string is that the initial losses resulted from an attempt, on the part of the machine, to imitate the opponent. Since the machine at first imitates only the opponent's position in a hopeless configuration, the resulting moves are at times startlingly unintelligent, and a certain amount of "relearning" is necessary.

2.3 An Alternate Scheme

It is interesting to contrast the performance of the machine with an earlier version whose four first move selection rules were based on the conditions:

1. $M_iC = TC_w$,

2. $TO_h \subseteq M_i'O$,

3. $M_iX \subseteq TX_w$,

4. $M_iX \subseteq TO_h'$,

and was otherwise identical with the final version. It should be noted that this earlier structure is much more defensive, since the blocking rule takes precedence over everything but a perfect match of a previous win. As a result, the earlier machine had a decided tendency to draw, and seldom took advantage of a possibility of winning if it could block the opponent instead. The final version is a much more interesting opponent but is liable to take a few more losses in its attempts to win. In return, its tendency to be satisfied with a draw is far less pronounced. However, it cannot be guaranteed that it will always try to win.

The last remark may be used as a criticism of our design. However, as will be shown in our second note, the defect can easily be corrected by providing more memory records. Owing to the limitations of the computer at our disposal, however, we were forced to attempt a more economical scheme.

An interesting distinction between the two machines was discovered when playing the game of Hex. The early and more defensive machine arrived at an unbeatable strategy after seven plays, machine starting, all of which were lost. The final machine had only a partial strategy after some 50 plays of which 41 were lost. The reason for this is that the asymmetry of the game makes the rule for imitating the opponent worthless, and hence results in a large number of futile attempts to "learn" by that rule. At the same time, since the game has only two values, a good defense is a reliable winning strategy. We have, unfortunately, not had the opportunity to carry the Hex-playing experiments to a conclusion.

2.4 Detection of Symmetries

The Hex-playing referred to above was done by initially informing the machine of the (six) invalid symmetries of the square. An attempt (with the early version of the machine) to play Hex without this initial information was carried out for about 100 plays with little success. At the end of the run, the machine memory showed that only one invalid symmetry had been detected.

This result was not quite unexpected. It should be noted that the determination of inadmissible moves involves the use of symmetries. Hence, when the game is strongly asymmetric, it may be expected that moves will be deemed inadmissible that in reality are good. In order to avoid this difficulty, it would have been advisable to introduce two categories of hopeless configurations and two admissibility criteria, one depending on the full group of symmetries, the other depending on no symmetries at all. If no admissible move existed under the first criterion, the machine would then attempt to find moves under the second criterion. This would, of course, not significantly affect its behavior in a symmetric game, but would avoid the difficulties in an asymmetric game. Since the machine would then "learn" to win, it would also detect invalid symmetries by the rules given for our version. The main difficulty with the present version, in addition to that already mentioned, is that so few wins occur early in the game that the machine has little or no occasion to test the symmetries. We have not had an opportunity to incorporate the above suggestions into our program.

Note II: On a Definition of Machine Learning

1. HEURISTICS

We consider automata that interact with their environments through the receipt and transmission of finite sequences of input and output symbols drawn from finite alphabets. It is common to define "learning" by postulating an observer who evaluates the interaction and communicates the result of his evaluation to the automaton. The latter is said to "learn" if, on the basis of the received evaluations, it proceeds to modify its responses so as to improve the evaluations with time. This definition, apart from being too vague, is totally useless since, as has often been remarked, it recognizes as "learning" the behavior of an automaton designed to respond perfectly and to ignore the evaluation signals. (The counterexample is sometimes given in terms of an automaton which simulates improvement, but this is, clearly, a superfluous nicety.)

The framers of such definitions are saved by the fact that they do not use them for other than rhetorical purposes—the question whether a particular automaton "learns" being decided by other, usually structural, criteria.

The difficulty discussed is easily overcome, however, by requiring that the automaton behave in the prescribed fashion for at least two incompatible evaluation schemes. This clearly disposes of the undesired cases since, although it is perfectly possible to design an automaton to respond perfectly to both evaluations, it must "learn" by experience which scheme is used by the observer. Note that the observer is not obligated to choose his evaluation scheme until after the automaton has made a response. Thus the modified definition may admit quite *trivial* instances, but this is not an unusual or objectionable feature.

There are, however, several items that require clarification before the modified definition becomes acceptable. Regarding the modification itself, it will be necessary to specify what is meant by "incompatible evaluation schemes." It is clearly not sufficient that the schemes be different, as the following example will show. Consider a scheme with only two values. A different scheme may be obtained by changing the value of an unfavorable response. An automaton that always responds favorably under the first scheme will do so under the second scheme as well.

A second item in need of clarification is the notion of improvement. We replace this by an upper bound on the number of times a mistake may be repeated. This may appear to be too strong a condition, but we feel that it is justified by our solitary theorem (in Section 5) concerning the realizability of automata that satisfy the condition.

Finally, it will be necessary to give a precise characterization of the notion of an evaluation scheme.

2. FORMAL PRELIMINARIES

Our definitions are to be framed in terms of the interaction between the automaton and its environment. It is therefore convenient first to develop a terminology with which to describe this interaction.

We assume given the input and output alphabets of an automaton. An *exchange*, c, is a finite sequence of input symbols followed by a finite sequence of output symbols. An *e-string* is a (finite or infinite) sequence of exchanges.

Let ϵ_1 and ϵ_2 be *e*-strings. We say that ϵ_1 is a *substring* of ϵ_2, or that ϵ_2 contains ϵ_1, if ϵ_1 is identical with an initial segment of ϵ_2. (The substring, ϵ_1, is *proper* if $\epsilon_1 \neq \epsilon_2$.) The *intersection* of ϵ_1 and ϵ_2 is the longest substring common to both. If the intersection of ϵ_1 and ϵ_2 is empty, the strings are said to be *disjoint.*

Two *e*-strings are said to be *output alternatives* if, upon deletion of the intersection from each string, the initial exchanges of the remainders have identical input sequences. An output alternative, ϵ_1, of ϵ_2 is said to be *minimal* if the remainder of ϵ_1 after deletion of the intersection of ϵ_1 and ϵ_2 is a single exchange.

Let E and E^* be two sets of *e*-strings. E is said to be *complete in E^** if every element of E^* contains an element of E, and every element of E is contained in an element of E^*.

A set of *e*-strings is said to be *proper for evaluation* if no element of the set is a proper substring of any other element of the set.

3. EVALUATIONS, ADAPTABILITY

It may be recalled that we are confining our attention to interactions in which finite input sequences always produce finite output sequences. Such interactions may be characterized by a set, E^*, of all *e*-strings that can occur starting with the automaton in any state, and continuing as long as the automaton will respond. It should be emphasized that the set, E^*, depends not only on the automaton in question but also on the environment. Hence the interaction of the same automaton with two different environments may be characterized by different sets, E^*.

In what follows we shall assume that a set E^* is given, and confine our discussion to *e*-strings that are substrings of elements of E^*.

An *evaluation* over E^* is an ordered, finite class of disjoint sets of finite *e*-strings whose union is complete in E^* and proper for evaluation. (The order of the sets is intended to reflect the intuitive order of increasing preference.)

It should be noted that an evaluation assigns an order relation to those *e*-strings that occur in its members. This order relation may be extended to all substrings of these *e*-strings in the following manner:

Let A be the set of all *e*-strings occurring in the members of an evaluation and their substrings. Let E be a member of the evaluation.

An *e*-string, ϵ, in A is said to be *superior* to E if (a) ϵ belongs to a set of higher order than that of E, or (b) every *e*-string in A that contains

ϵ is, or has a minimal output alternative containing ϵ that is, superior to E.

The *order* of an e-string, ϵ, in A is that of the lowest order set to which ϵ is not superior.

Let E be a member of an evaluation, and let ϵ be an element of E. We shall say that ϵ is a *mistake* if it has a minimal output alternative that is superior to E.

We now stipulate that, for a given automaton, there shall be specified a fixed, ordered set of input sequences, to be designated as *evaluation inputs*, into which an evaluation shall be mapped by mapping the ith highest order member of the evaluation on the ith sequence of the set.

By a *normal* L-*situation* we shall mean a situation in which the interaction of an automaton with its environment is characterized by a set, E^*, subject to the following, additional stipulation: Counting from the initial state, as soon as the interaction has produced an e-string occurring in a member of a given evaluation, the corresponding evaluation input shall be presented to the automaton. This procedure shall then be repeated throughout the interaction, taking the state following each evaluation input as a new "initial state."

It should be observed that the completeness requirement for an evaluation guarantees that any interaction will give rise to an evaluation input after a finite number of exchanges.

An automaton is said to be *adaptable* to a given evaluation if, under a normal L-situation, there exists, for each mistake, a finite upper bound on the number of possible occurrences of that mistake.

4. L-AUTOMATA AND L-BEHAVIOR

Two evaluations over the same set E^* are said to be *incompatible* if each evaluation determines a set of e-strings with the following properties.

(a) All e-strings are of the same order, r (say).

(b) If ϵ is in the set and ϵ' is a minimum output alternative of ϵ of order r, then ϵ' is contained in some element of the set.

(c) No minimum output alternative ϵ' is of order greater than r.

(d) With respect to the *other* evaluation, if ϵ is in the set, there exists a minimal output alternative of ϵ whose order exceeds that of ϵ.

A consequence of this definition is that an automaton that responds perfectly to one of two incompatible evaluations can always be forced to make a mistake with respect to the other.

We are now in a position to give our main definition:

Let the interaction of an automaton and an environment be characterized by a set, E^*, of e-strings. The automaton is said to be an L-*automaton*, and its behavior is referred to as L-*behavior*, if it is adaptable to at least two incompatible evaluations over E^*.

5. REALIZABILITY

We shall say that the interaction between an automaton and its environment is *bounded* if there exists a finite upper bound on the lengths of the input and output sequences that can occur. Our objective in this section is to show that the design problem for automata whose interaction is bounded is solvable.

Let the *length* of an e-string be the number of exchanges occurring in it. The *ϵ-order* of an evaluation is the length of the longest e-string in the union of its members. We have the following theorem.

THEOREM. *Given a set, E^*, characterizing a bounded interaction, there exists, for any positive integer, N, an* L-*automaton that is adaptable to all evaluations over E^* of ϵ-order not exceeding N.*

The validity of this theorem will be established by showing how such an L-automaton may be programmed on a computer. No formal proof is offered.

By the *input configuration* to the computer at any time we shall mean the e-string representing the interaction following the last evaluation input, and the present input. The computer output, given an input configuration, shall be determined as follows.

An output sequence shall be called *inadmissible* if the addition of the output sequence to the input configuration produces an e-string listed in memory. Otherwise the sequence shall be called *admissible.*

Given an input configuration, the computer shall respond with the alphabetically first of the admissible sequences. If no admissible sequence exists, the response shall be the alphabetically first of those sequences that produce the e-strings listed with highest evaluation.

Memory records shall be made according to the following rules.

An e-string receiving the highest possible evaluation shall be disregarded. An e-string receiving a lower evaluation shall be listed, together with its evaluation, the first time it occurs. If an input configuration admits of no admissible sequence, then the e-string belonging to that configuration shall be listed with the evaluation belonging to the output selected. The evaluation of such an entry may be lowered at a later stage.

We claim that, for any evaluation, the above program will produce adaptation in the sense defined previously. Because of the finite character of the situation, only a finite number of entries are required, and only a finite number of output sequences need to be tested for each input configuration. Hence the program is realizable.

RECEIVED: February 3, 1959; Revised: June 3, 1959.