

# SELF-ORGANIZATION: Meaning and Means

*George G. Lendaris and Gordon L. Stanley*  
*GM Defense Research Laboratories*  
*Santa Barbara, California*

## INTRODUCTION

As outlined by the chairman, this session of the Second Congress is concerned with the formal, theoretical problems of self-organizing systems. Accordingly, this paper discusses in a formal way the ideas of system, self-organizing system, constraint, and generalization as applied to such systems (Section I), and presents some theoretical (and experimental) results toward the development of a self-organizing machine (Section II). These results were derived using the Ashenhurst-Curtis theory of decomposition of switching functions.<sup>1,2</sup> The authors feel that this theory should be exploited as a tool in the development of self-organizing systems.

The definitions of Section I are intended to formalize some of the notions that are being employed in the study of self-organizing systems. In addition, the notion of generalization is treated from a quantitative point of view instead of the usual qualitative one. Some ideas important to this quantitative approach are developed, with the result that the generalizing ability of several self-organizing systems can be compared quantitatively. The formal definitions of Section I are not necessary to the developments of Section II.

In Section II, a teacher-pupil attribute is assigned to a self-organizing system; the pupil is assumed to be a collection of variable logic elements. Most published results have been for the case where the variable logic elements of the pupil are noninteracting. When the elements of the pupil are allowed to interact, the rules (or algorithm) by which the teacher trains the pupil become increasingly diffi-

cult to develop. Thus, one important result of this paper is Training Algorithm 1 (and its attendant proof of convergence) which solves this problem for two interacting elements. The greatest difficulty in teaching a pupil with *many* interacting variable elements is in preserving good information after the pupil once acquires it. Training Algorithms 2 and 3, based on Algorithm 1, solve this problem for a specific many-element pupil, called a LADICAN.

The structure (interconnection of elements) of a pupil determines precisely the set of tasks (functions) it can perform. It is thus important to investigate the structure-dependent properties of the many-element pupil; several of these properties have been determined for the LADICAN. The Ashenhurst-Curtis theory was the basic tool employed in the developments discussed in this section.

Experimental results were obtained by simulating LADICANs on a digital computer. Some of these results are presented.

## SECTION I—MEANING

### 1.1 *System and Self-Organizing System*

This section will be concerned with presenting a definition of self-organizing system and then extracting from this definition knowledge of the structure of a self-organizing system, to be given as a theorem.

To define a self-organizing system, the term, system, must first be defined. Abstractly, for an entity to merit the name system (as generally applied) it must in some way process information. That is, starting with some kind of "inputs," it performs

some operations on these inputs, and yields the consequences of these operations (called "outputs")—this is all that is necessary to a system. This applies, for example, to a simple R-C network, to a radio containing the network as a component, or to an entire broadcasting system containing both network and radio as components. In each case, the system (or respective component-system) has what may be called parameters which affect the way in which the system operates (for example, in the R-C network, the values of R and C affect its operating characteristics; and in the radio, the volume and tuning controls affect the way the radio system operates—each affecting the operation of the system in which it is contained).

With this discussion as a motivation, the following sequence of definitions and axioms is presented.

**Definition 1:** A *system*  $S$  is a triplet:  $S = [I, O, R]$ , where  $I = I_r X I_p$  (Cartesian product of  $I_r$  and  $I_p$ ).  $I_r = \{U_s(t)\}$ , the set of all  $U_s(t)$  where  $s$  runs over some index set;  $U_s(t)$  is a vector time function defined on, say,  $t \geq 0$ ; and the components of  $U_s(t)$  are the *Regular-inputs* of  $S$ .  $I_p = \{V_s(t)\}$ , the set of all  $V_s(t)$  where  $s$  runs over some index set;  $V_s(t)$  is a vector time function defined on, say,  $t \geq 0$ ; and the components of  $V_s(t)$  are the *Parameter-inputs* of  $S$ .  $O = \{C_s(t)\}$ , the set of all possible  $C_s(t)$  where  $s$  runs over some index set;  $C_s(t)$  is a vector time function defined on, say,  $t \geq 0$ ; and the components of  $C_s(t)$  are the *Outputs* of  $S$ .  $R$  is a relation between  $I_r$  and  $O$ , i.e., a subset of the Cartesian product  $I_r X O$ , and  $R$  is dependent upon the parameter-inputs. That is,  $R = F(V_s)$ , where  $F$  is a function from  $I_p$  to  $B$ , where  $B = \{R | R \subset I_r X O\}$ , the set of all possible relations between  $I_r$  and  $O$ . If  $R$  is fixed, the system is said to be fixed. If  $R$  is variable, the system is said to be variable. (A variation in  $R$  is effected by a variation in a parameter-input.) A system  $S$  can be composed of other systems, say  $S_1$  and  $S_2$ , each defined by its respective triplet,  $S_1 = [I_1, O_1, R_1]$  and  $S_2 = [I_2, O_2, R_2]$ .  $S_1$  and  $S_2$  will be called *Component-systems*.

Now, partition the set of parameter-inputs into two subsets: let  $\Gamma$  be the set of those parameter-inputs which will be called "intentional," and  $\Lambda$  the set of all remaining parameter-inputs. Let  $\gamma$  be a subset of  $\Gamma$  and  $\lambda$  a subset of  $\Lambda$ .

**Definition 2:** A system  $A$  is said to *dominate* a system  $B$  when an output of system  $A$  is a parameter-input of system  $B$ , and no output of  $B$  is a parameter-input of  $A$ .

**Definition 3:** A *system self-organizing with respect to a relation*  $D$ , call it  $S_D$ , is a 4-tuple:  $S_D = [I, O,$

$R, D]$ , where  $R$  goes to  $D$  ( $R \rightarrow D$ ) with time, and there are no  $\Gamma$ -parameter-inputs to  $S_D$ .  $I, O, R$  are as in Definition 1.  $D$  is a specified relation between  $I_r$  and  $O$ .\*

As a first step, the necessary axiomatic assumptions regarding the existence of  $S$  and  $S_D$  are made. Assuming that an  $S_D$  exists, then by Definition 3,  $R \rightarrow D$  with time; hence it follows that  $R$  is variable (except for the trivial case where  $R = D$ ).

Since  $R \rightarrow D$  with time, it is reasonable to assume that  $R$  varies purposefully (in some sense) at least part of the time, because it does not seem probable that  $R \rightarrow D$  solely by random changes in the parameters. With this argument as motivation, the following axiom is stated.

**Axiom 1:**

$(R \rightarrow D \text{ with time}) \rightarrow (R = R(\gamma, \lambda))$ , where  $\gamma \neq \phi$  and  $R$  is not a constant function of  $\gamma$

Axiom 1 requires the existence of some parameter-inputs which are changed intentionally (i.e.,  $\gamma \neq \phi$ ). Definition 3 asserts that there are no such parameter-inputs to  $S_D$ ; therefore, there must be a source of the  $\gamma$  within  $S_D$ , that is, a component-system (call it  $W$ ) within  $S_D$  with outputs  $\gamma$  (viz.,  $W = [I_w, \gamma, R_w]$ ). Since these parameters serve to change  $R$ , component-system  $W$  must dominate some other component-system (call it  $V$ ) within  $S_D$ —one that is characterized by this  $R$  (viz.,  $V = [I_v, O, R(\gamma, \lambda)]$ , where  $R$  is the same relation as that for  $S_D$ ); therefore, the regular inputs to  $V$  are the same as those to  $S_D$ ; i.e.,  $I_v = I_r X I_{p_v}$ . Furthermore, the system  $W = [I_w, \gamma, R_w]$  must be such that  $R(\gamma, \lambda) \rightarrow D$ .

These considerations lead to the conclusion that  $S_D$  must have a component-system of the form  $V = [I_v, O, R(\gamma, \lambda)]$ ,  $\gamma \neq \phi$ , and a component-system of the form  $W = [I_w, \gamma, R_w]$ , where  $W$  dominates  $V$  in such a way that  $R(\gamma, \lambda) \rightarrow D$ . (By this last requirement,  $W$  in essence represents  $D$ .) The requirement that  $W$  causes  $R \rightarrow D$  implies that  $W$  receives some information regarding  $R$ ; since  $R$  is a relation between  $I_r$  and  $O$ , this means that  $W$  receives as its regular-inputs at least some of the regular-inputs of  $V$  and at least some of the outputs of  $V$ .

These deduced necessary requirements are represented in Figure 1 via standard block diagram notation. This block diagram represents a minimum necessary interconnection pattern for a system to be self-organizing with respect to a relation  $D$  as

\*  $[I, O, R, D] \rightarrow [I, O, D, D] \equiv [I, O, D]$

1.2 Constraint

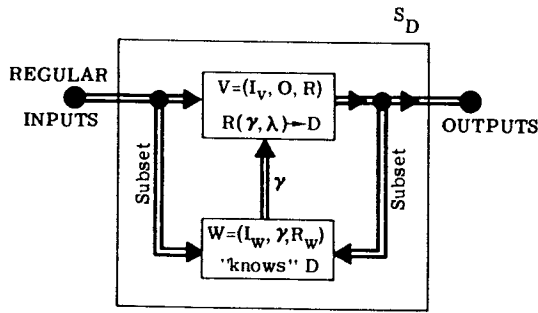


Figure 1. A Minimum Necessary Interconnection Pattern for a System to be Self-Organizing With Respect to a Relation  $D$ , as Defined in Definition 3.

defined in Definition 3. The word "subset" is used in the diagram to indicate that at least some, or possibly all, of the respective information is included in the labeled path. No attempt was made to include the unintentional-parameter-inputs, since they can come from anywhere.

In practice, determination of which inputs to the system are to be called regular-inputs and which ones are to be called parameter-inputs is determined by  $D$ . This is because  $D$  expresses a relation between certain inputs and the outputs; thus in specifying  $D$ , a set of inputs is specified. These latter inputs are the regular-inputs and all others the parameter-inputs.

For the converse of the above arguments, if a system  $S = [I, O, R]$  has a component-system of the form  $V = [I_v, O, R(\gamma, \lambda)]$  and a component-system of the form  $W = [I_w, \gamma, R_w]$ , where  $W$  dominates  $V$  in such a way that  $R(\gamma, \lambda) \rightarrow D$ , where  $D$  is a specified relation between  $I_r$  and  $O$ , then  $S$  is self-organizing with respect to  $D$ . This follows from the Definition 3. Thus, an alternate definition of a self-organizing system is given by the following theorem.

**Theorem.** A system  $S = [I, O, R]$  is self-organizing with respect to a specified relation  $D$  between  $I_r$  and  $O$  if, and only if,  $S$  has a component-system of the form  $V = [I_v, O, R(\gamma, \lambda)]$ ,  $I_v = I_r \times I_{p_v}$  and a component-system of the form  $W = [I_w, \gamma, R_w]$ , where  $W$  dominates  $V$  in such a way that  $R(\gamma, \lambda) \rightarrow D$  with time.

The forte of this latter definition, over Definition 3, is its display of some (minimum necessary) properties of the structure of a self-organizing system.

The above discussion is essentially the same as that of Reference 3.

The remaining discussions of this section are directly concerned with regular inputs only—not parameter-inputs. Thus, when the word "input" appears, it means regular input.

Given a function  $F(x_1, x_2, \dots, x_n)$ . Each independent variable,  $x_i$ , ranges over a set of values  $X_i$ . If a value is assigned to  $F$  for each  $n$ -tuple  $(x_1, x_2, \dots, x_n)$  with  $x_i \in X_i$ , then  $F$  is said to be a *Total Function*; if there are some joint values of  $x_1, x_2, \dots, x_n$  for which  $F$  is undefined,  $F$  is called a *Partial Function*.

We will call those  $n$ -tuples for which  $F$  is defined *care terms*, and those  $n$ -tuples for which  $F$  is undefined, *don't-care terms*. We will use the word *constraint* to indicate that not all  $n$ -tuples  $(x_1, x_2, \dots, x_n)$  with  $x_i \in X_i$  are of interest. Thus, in a constrained input set, there is a non-empty set of don't care terms.

In this paper, we restrict our attention to the finite case. That is, the sets  $X_i$  will be finite. However, much of what is said applies directly to the non-finite case.

We assume the relation  $D$  of  $S_D$  to be a function. As such, it assigns one, and only one, output to an input. If  $D$  specifies an output for every possible input, it is total; otherwise, it is partial.

To illustrate, consider the following problem: on a square grid of  $m$  black/white squares, distinguish between all patterns considered to look like  $A$ 's and all considered to look like  $B$ 's. On this square grid, there are  $2^m$  possible patterns ( $m$ -tuples). The independent variables  $x_1, x_2, \dots, x_m$  range over the set  $(0, 1)$ . The set of care terms is constrained to those  $m$ -tuples corresponding to  $A$ 's and  $B$ 's; the balance are don't-care terms. Zero is to be assigned to each  $n$ -tuple corresponding to an  $A$ , and 1 to each  $m$ -tuple corresponding to a  $B$ ; no output is defined for the don't-care terms. This specification is a partial function. (We will refer to this example later).

Given two functions,  $F$  and  $G$ , with the same independent variables. If for every  $n$ -tuple  $(x_1, x_2, \dots, x_n)$  for which  $F$  is defined,  $G$  is also defined and assigns the same value as  $F$ , then  $G$  is an *extension* of  $F$ . The notation used to denote this is  $F \subseteq G$ . (Alternatively, this notation may be read:  $F$  is a *restriction* of  $G$ .)

Assume a self-organizing system  $S_D = (I, O, R, D)$  as in Fig. 1. For any input to  $S_D$ ,  $V$  will issue an output. Thus  $R$  is total. Letting  $D$  be a partial function, we assume that the inputs to  $S_D$  are only the care terms of  $D$ , and these we call set  $C$ . Let  $S_D$

operate until  $V$  issues the correct output for each of the inputs in set  $C$ .  $R$  will then be an extension of  $D$ ,  $D \subseteq R$ .

Next, assume another self-organizing system  $S_{D_1} = [I, O, R_1, D_1]$  where  $D_1 \subseteq D$ , and the set of care terms of  $D_1$ , call it  $C_1$ , is a proper subset of  $C$ , denoted  $C_1 \subset C$ . Let  $S_{D_1}$  operate until  $D_1 \subseteq R_1$ .

In terms of the previous example,  $C$  is the entire set of  $A$ 's and  $B$ 's; and when  $S_D$  is finished,  $V$  correctly identifies all  $A$ 's and  $B$ 's. Since  $C_1$  is only some subset of  $C$ , all that can be said when  $S_{D_1}$  is finished is that  $V_1$  correctly identifies the letters  $A$  and  $B$  belonging to  $C_1$ .

Now let us compare  $R_1$  and  $D$ . That is, with  $W_1$  disconnected, how well will  $V_1$  identify the  $A$ 's and  $B$ 's it has not "seen" (those belonging to  $C/C_1$ )? If  $D \subseteq R_1$ ; then, and only then, will  $V_1$  correctly identify all the  $A$ 's and  $B$ 's. At the other extreme, if  $D$  and  $R_1$  have no assignments in common for the care terms in  $C/C_1$ , then  $V_1$  will make a "mistake" on all the letters  $A$  and  $B$  it has not "seen."

At this point, a semantic problem arises. How do we use one word, or one concise set of words, to describe where we are in this range of possibilities? The word "generalization" has been used in this context. For example, if, when tested on a few care terms from  $C/C_1$ ,  $V_1$  yields the correct responses, then  $V_1$  has been said to generalize. The usage to date seems quite imprecise.

### 1.3 Generalization

The dictionary definition of generalization contains no assertion of "correctness." Therefore, any *extension* of a function is a *generalization* of that function. The problem that remains is to compare this generalization with some other function, partial or total, to determine the degree of concurrence, or "correctness," over the set of care terms. No matter what the application, a comparison with some standard must be made before an assertion of "correctness" is made.

Using the previous notation, assume partial function  $D$  as our standard. We have  $D_1 \subseteq D$  with  $C_1 \subseteq C$ , and  $D_1 \subseteq R_1$ . As stated earlier, we wish to compare  $D$  and  $R_1$ . (We recall that all these functions have the same independent variables.) The functions  $D$  and  $R_1$  give the same assignments to the care terms in set  $C_1$ . Since  $D$  is defined only over  $C$ , we are not interested in the assignments  $R_1$  makes over  $I_r/C$ . (Recall that  $I_r$  is the set of all

possible inputs.) The comparison is thus to be made only over the set of care terms  $C/C_1$ .

Let the cardinality of  $C$  be  $c$ , and of  $C_1$ ,  $c_1$ . The cardinality of  $C/C_1$  is thus  $(c-c_1) \triangleq k$ . Let  $k_c$  be the number of care terms in  $C/C_1$  for which  $R_1$  assigns the correct value (the same value that  $D$  assigns). We use the ratio  $k_c/k$  as an adjectival modifier for the word "generalization" to indicate the relative degree of concurrence of  $R_1$  with  $D$  over the care terms  $C/C_1$ —that is, the degree of correctness.

Any extension, or generalization, is made relative to a base set of care terms—in this case set  $C_1$ . The care terms of  $C_1$  are those care terms of  $C$  to which  $V_1$  is exposed. Thus the ratio  $c_1/c$  in a sense tells us how much we pay for the generalization we get.

We incorporate these two ratios in the following expression.

$$\frac{k_c}{k} \text{ generalization via } \frac{c_1}{c} \text{ exposure} \quad (1)$$

If  $k = 0$ , define  $\frac{k_c}{k} = 0$ .

As an example, suppose  $c = 100$  and  $c_1 = 50$ ; then  $k = 50$ . Suppose  $k_c = 25$ . For this case we have

$$.5 \text{ generalization via } .5 \text{ exposure}$$

As another example, again let  $c = 100$  but let  $c_1 = 24$ ; then  $k = 76$ . Suppose  $k_c = 38$ . For this case we have.

$$.5 \text{ generalization via } .24 \text{ exposure.}$$

The .5 generalization in the latter case is more impressive than in the former.

Though its intent may be precise enough, Expression (1) has its practical difficulties. In most applications,  $c$  and  $k$  are very large numbers. In these cases it is impractical to compare  $R_1$  and  $D$  at each of the  $k$  care terms not in the base set.

For practical considerations, therefore, a compromise must be made.

Instead of using the entire set  $C/C_1$ , we select some representative members of  $C/C_1$ , and examine  $R_1$  with these care terms, called the *exam set*; we denote the exam set by  $E$ , and its cardinality by  $e$ . We let  $e_c$  be the number of care terms in the exam set  $E$  for which  $R_1$  assigns the correct value. The ratio  $e_c/e$  will then replace the ratio  $k_c/k$  in Expression (1).

$$\text{We modify the ratio } c_1/c \text{ to be } \frac{c_1}{c_1 + e}$$

This is the ratio of the size of the base set to the total number of care terms to which  $V$  is exposed.

\*  $C/C_2 = \{y \mid y \in C \text{ and } y \notin C_2\}$

When  $e = k, \frac{c_1}{c_1 + e}$  becomes  $\frac{c_1}{c}$ .

For the application to be discussed later, system  $V$  is assigned the role of pupil, and system  $W$  is assigned the role of teacher. We then speak in terms of  $W$  training  $V$  to perform  $D$ . The teacher trains the pupil with a subset of care terms, called the *training set*,  $T$ , of cardinality  $t$ . ( $T$  corresponds to the set  $C_1$  in the above discussion.) After the teacher has trained the pupil, we examine the pupil to see how well it performs. The exam set is selected from  $C/T$ .

It may turn out that the pupil does not give correct answers for all the care terms in the training set,  $T$ . This must be taken into account when reporting the results of the pupil's performance on the exam set.

Let  $t_c$  be the number of care terms in  $T$  for which the pupil does give the correct output. The ratio  $t_c/t$  will be used as a measure of the pupil's performance on the training set.

We propose the following expression to encompass all the above considerations:

$$\frac{e_c}{e} \text{ generalization via } \frac{t_c}{t} \text{ training performance on} \\ \frac{t}{t + e} \text{ exposure} \quad (2)$$

It is important to remember that this statement depends on the exam set  $E$ . An a priori judgment has to be made that the set  $E$  is representative of the set  $C/T$ .

#### 1.4 Generalization Ratio

We believe that the greatest potential attribute of a self-organizing machine will be its ability to generalize well. Therefore, in developing self-organizing machines, we will want to compare the generalization performance of one teacher-pupil combination on a training and exam set with that of another combination on the same sets. Or, we may want to compare the difference in performance of just one teacher-pupil combination on several training and exam sets.

As an example, suppose pupil 1 has .8 generalization via .8 training performance on a given exposure whereas pupil 2 has .8 generalization via 1.0 training performance on the same training exposure. Which pupil did "better" at generalizing? We assert that pupil 1 did, because it performed as well on the care terms to which it had never been exposed as it did on the care terms of the training set. Pupil 2, how-

ever, performed less well on the exam set than on the training set.

As an index of a pupil's generalizing capability, we define a *Generalizing Ratio*:

$$GR = \frac{e_c/e}{t_c/t}$$

When two Generalization Ratios are compared, it is assumed that either the sets  $T$  and  $E$  are held constant with variations occurring in the pupil and/or teacher, or that the pupil-teacher combination is fixed with variations occurring in the selection of sets  $T$  and/or  $E$  for constant  $\frac{t}{t + e}$ .

For the above illustration,

$$GR (\text{pupil 1}) = 1.0$$

and

$$GR (\text{pupil 2}) = .8$$

This ratio indicates that pupil 1 did better at generalizing than pupil 2.\*

## SECTION II—MEANS

### 2.1 Introduction

This section is concerned with aspects of developing a self-organizing machine.

Many of the results to be presented here have been derived in Reference 4. The derivations are all based upon the Ashenurst-Curtis theory of decomposition of switching functions,<sup>1,2</sup> hereafter referred to as the A-C theory. In Reference 4, the portions of the A-C theory necessary to the development are presented.†

In this section the word *machine* represents an embodiment of the concept of *system*, presented in Section I. The kind of machine considered is as follows: A set of binary inputs is presented to the machine; the machine yields a set of binary outputs. The machine has two parts: a teacher, and a pupil. The teacher "knows" what response is desired from the pupil and trains the pupil to achieve this desired performance. The pupil is a network (or net) of interconnected binary logic elements, some of whose parameters are variable. The *structure* of the network is the interconnection pattern of the elements.

This machine clearly satisfies the conditions of Theorem 1 and may, therefore, be called self-organizing with respect to the function the teacher "knows."

\* The ideas of Sections 1.3 and 1.4 are discussed less formally in Reference 14.

† Reference 4 is self-contained and is tutorially oriented.

The success of the teacher in training the pupil will depend upon the use of an appropriate training algorithm. As presently visualized, the teacher's modus operandi consists of two major phases: adjusting and restructuring. In the adjusting phase, the parameters of individual logic elements comprising the network are changed, but the interconnection among those elements is not. In the restructuring phase, the interconnection among the elements is changed. In these terms, the process of teaching proceeds by starting from an initial network; there is an adjustment phase during which an attempt is made to have the pupil learn an assigned function. This phase ends when either the function is learned, or the teacher decides that the function is beyond the pupil's capability. If this happens, a restructuring phase begins, and interconnections among the elements are modified. Then a new adjusting phase is started.

With the A-C theory, it can be shown that in any specifically structured network of logic elements, the given structure of the network determines precisely the set of functions realizable by it. This set of realizable functions is a subset of the set of all possible functions of the given input variables. Further, this subset of realizable functions is quite independent of the algorithm used by the teacher in training the pupil.\*

We define the *performance space* of a network as the set of functions that the pupil is capable of performing as the element parameters range over their possibilities but the structure (interconnection of elements) is unchanged. Each structure has its corresponding performance space. In these terms, during the adjusting phase a scheme of parameter adjustment is followed to seek out solution values if the function is within the pupil's performance space. If the function is not within the pupil's performance space, this is to be determined and the performance space meaningfully changed by restructuring the elements.

An illustration of these ideas is given by Figure 2. The total number of possible switching functions that can be performed on  $m$  binary variables is  $2^{2^m}$

\* Therefore to ask whether or not a given machine is self-organizing with respect to a certain function is meaningful only if that function is contained in the set of functions realizable by the pupil. When such a function is specified to the teacher and the pupil learns it, the machine will be said to achieve the function. If for a given self-organizing machine the set of its achievable functions is equal to the set of its realizable functions, the machine is said to be "perfect." Thus, if the training algorithm forces the pupil to converge to a solution if it exists, the machine will be "perfect."

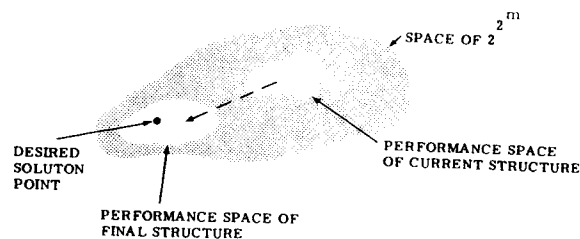


Figure 2. Performance Space Directed to Contain Desired Solution Point.

The performance space of a structured network (with  $m$  external inputs) is some subset of these  $2^{2^m}$  functions. The desired function may or may not be contained in the performance space of the beginning structure. The object of the restructuring phase of the training algorithm is to change the structure in such a way that eventually the performance space will contain the solution function.

There are five crucial problems in this proposed operation:

1. Can a training algorithm be found that will assure convergence to a solution if it exists within the current capability of the pupil?
2. What is an appropriate starting network?
3. How is the performance to be monitored?
4. Upon what basis is the restructuring decision to be made?
5. What is a meaningful change?

Although these questions are not yet fully answered, they are listed here to serve as a reference frame for the results to be presented.

To answer these questions, considerable information regarding the structural properties of the networks will be required. To investigate the structural properties of these kinds of networks, a decision must first be made concerning the type of structure to start with. After the structure is chosen, some of the questions that will arise are the following:

1. How good is it; e.g., how many functions can it realize?
2. What is a training algorithm which will assure convergence for this net?
3. How can this training algorithm-net combination be meaningfully tested; e.g., can functions be specified with an a priori knowledge of the relative difficulty the pupil will have in learning them?

4. Assuming that the pupil has learned such a function in an unknown situation (such as in a pattern recognition task), is there anything that can be extracted from the resulting parameters of the net that will give information about the task and/or environment?

As a starting point, it was decided to use a cascade-type of network for the pupil because of the relative ease of applying the A-C theory, and to begin with a disjunctive partitioning on the inputs (each external input goes to only one element within the net). This decision was based upon the conjecture put forth by Curtis that if a function can be realized disjunctively, then this realization is probably minimal. Next, because of its generality—namely, that it can perform all possible switching functions of its input variables—the universal logic element was chosen as the basic component of the network. In this way, any network properties which are determined will be dependent upon the structure of the net, and not upon constraints imposed by the elements. By applying the A-C theory to disjunctive cascade networks, at least partial answers to the questions above were developed, and those pertaining to the first three questions are presented here.

### 2.2 Universal Logic Element

Consider a black box with  $m$  input wires, and one output wire; each wire corresponds to a binary variable. The box assigns one, and only one, output to each input. The operation of the box may be described as a switching function—a two-valued function of  $m$  two-valued variables. Let this black box have the added property that the switching function it performs on its binary inputs can be specified externally, say, by a dial setting. We define a *universal binary logic element* as such a black box with  $m$  inputs (denoted by  $U_m$ ) whose dial can assume any one of  $2^{2^m}$  positions, each position corresponding to a different one of the  $2^{2^m}$  switching functions that are possible on  $m$  binary inputs. A more restricted type of logic element can be obtained by suitably restricting the dial settings. In fact, all binary logic elements can be described in this way because any possible restriction of the operation of the element can be realized by an appropriate restriction on the dial settings. An example of such a restricted type of logic element is the threshold logic element. In this case, the set of permissible

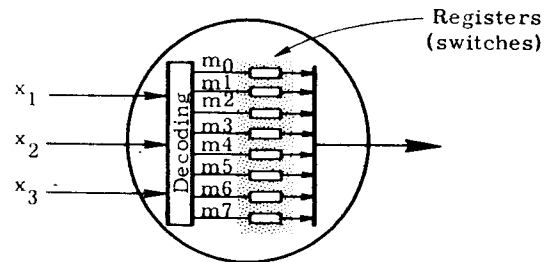


Figure 3. An Example of the Internal Structure of a Universal Element.

dial positions corresponds to those switching functions that are classified as linearly separable.

The universal logic element can be thought of as operating in the following manner. (See Fig. 3.) It has a first layer which performs a full decoding—that is, one which observes the  $m$  input wires, decides which pattern (minterm) is present, and announces this decision by sending a 1 out on the appropriate output wire (there are  $2^m$  output wires representing the  $2^m$  possible minterms of the  $m$  inputs). Following this first layer, there is a second layer which contains  $2^m$  registers, each one fed by one of the  $2^m$  minterm wires from layer 1. Although the precise operation of these registers depends on the specific application of the element, if the register contains a zero, its output will always be zero; but if the register contains a 1, its output will be the same as its input (i.e., the register may be thought of as a switch, open when zero, closed when 1). The outputs of all these registers feed an “inclusive or” gate whose output is the output of the element. Thus if the register (switch) that corresponds to the minterm being presented to the element is set at 1 (closed), the output of the element is 1; otherwise it is zero. Which registers are set to 1 (switches closed) depends upon the dial setting, i.e., the switching function to be performed by the logic element.

### 2.3 Number of Functions Realizable by a LADICAN, $R_n$

Consider a cascade structure of universal elements as shown in Fig. 4. As indicated, the number of external inputs to the  $j^{th}$  element is given by  $i_j$ .

We are specifically interested in the case where the external inputs to this network are partitioned into disjoint sets so that no input goes to more than one element within the net. With this restriction,

$$\sum_{j=1}^n i_j = m,$$

where  $m$  is the total number of external inputs. A switching function realizable by such a network is

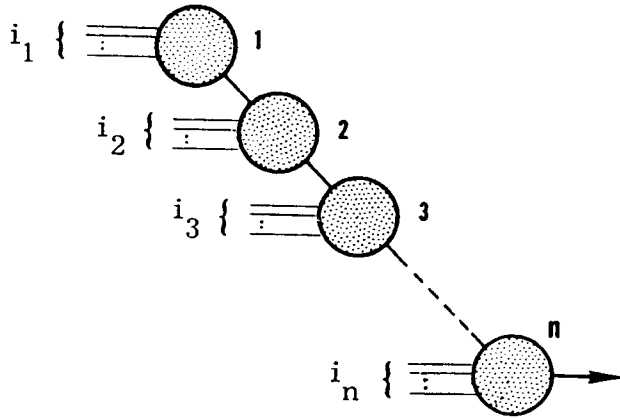


Figure 4. Labeled Disjunctive Cascade Network of  $n$  Universal Elements.

called disjunctively decomposable; therefore, we will call this network a labeled *disjunctive* cascade network, where the word *disjunctive* refers to the above restriction on the partitioning of the external inputs, and the word *labeled* means that a specific assignment of inputs has been made. For notational convenience, we will use the acronym LADICAN in place of *labeled disjunctive cascade network*. For further notational convenience, we let  $2^{i_j} = I_j$ , and we will let  $R_n$  be the number of functions realizable by a LADICAN of  $n$  universal elements. ( $R_n$ , of course, will depend upon the size of the universal elements.)

We state the following result:<sup>4</sup>

$$R_n = 2^{I_n} + 2^{I_{n-1}} (2^{I_n} - 1) (R_{n-1} - 2) \quad (3)$$

$$R_1 = 2^{I_1}, n = 2, 3, 4, \dots$$

This is the total number of functions realizable by a labeled disjunctive cascade network (LADICAN) of  $n$  universal elements.

For the special case where all the elements of the LADICAN have the same number of inputs, that is,  $i_1 = i_2 = \dots = i_n$ , we have a nonrecursive formula for  $R_n$ . We let  $I = 2^{i_j}, j = 1, 2, \dots, n$ . Then,

$$R_n = 2^I A^{n-1} - B \frac{A^{n-1} - 1}{A - 1} \quad (4)$$

where  $A = 2^{I-1} (2^I - 1)$

and  $B = 2^I (2^I - 2)$

For the special case where all the elements of the LADICAN are the same size, say  $U_k$ , we again have a nonrecursive formula for  $R_n$ :

$${}_k R_n = mc^n + p \quad (5)$$

where  $a = 2^{2^k}, b = 2^{2^{k-1}}, c = \frac{b}{2} (b - 1),$

$$d = b (b - 2) = 2c - b$$

$$p = \frac{d}{c - 1}, m = \frac{1}{c} (a - p)$$

and the subscript in front of  $R_n$  means that only  $U_k$  elements are in the LADICAN.

As a subcase of this last result, if the LADICAN contains only  $U_2$ 's, we have

$${}_2 R_n = \frac{12}{5} 6^n + \frac{8}{5} = \frac{2}{5} (6^{n+1} + 4) \quad (6)$$

Equation (6) was first presented in a paper by K. K. Maitra.<sup>6</sup> His derivation of this equation in that paper, based on a graphical scheme, failed to yield the generalization which was made in a straightforward manner in Equation (3) by using the theory of decomposition.

We note that for  $n = 2, 3, 4$ , the values of  ${}_2 R_n$  are respectively 88, 520, and 3112. These can be compared to the total number of switching functions possible on the corresponding number of inputs: 256; 65,536; 4,294,967,296. We note from these numbers that the "logical capacity"\* of a LADICAN decays very rapidly as  $n$  increases.

#### 2.4 Number of Ways of Realizing Functions in a LADICAN

The number of dial positions on a  $k$ -input universal element is  $2^{2^k}$ ; therefore, the total number of dial position combinations that can be made in a LADICAN of  $n$  universal elements is  $2^{2^{i_1}} 2^{2^{i_2}+1} \dots 2^{2^{i_n}+1}$  (Refer to Fig. 4). This number is,

\* Adapting a definition given by Maitra, "logical capacity" in this case is the ratio of

$$R_n / \left( \prod_{j=1}^n 2^{i_j} \right) \text{ where } \prod_{j=1}^n 2^{i_j}$$

is the total number of possible switching functions on the

$$\sum_{j=1}^n i_j \text{ inputs.}$$

Subset	Number of Realizations	Cardinality of Subset
0	$2^{n-1} T_0$	$(2^{I_1-2})2^{I_2-1} (2^{I_2-1}) \dots 2^{I_n-1} (2^{I_n-1})$
1	$2^{n-2} T_1$	$(2^{\hat{I}_2-2})2^{I_3-1} (2^{I_3-1}) \dots 2^{I_n-1} (2^{I_n-1})$
2	$2^{n-3} T_2$	$(2^{I_3-2})2^{I_4-1} (2^{I_4-1}) \dots 2^{I_n-1} (2^{I_n-1})$
.	.	.
.	.	.
.	.	.
n-3	$2^2 T_{n-3}$	$(2^{I_{n-2}-2})2^{I_{n-1}-1} (2^{I_{n-1}-1})2^{I_n-1} (2^{I_n-1})$
n-2	$2 T_{n-2}$	$(2^{I_{n-1}-2})2^{I_n-1} (2^{I_n-1})$
n-1	$T_{n-1}$	$2^{I_n}$

where  $T_j = 2T_{j-1} (2^{I_j+1} - 1) + 2^{I_1} 2^{2I_2} \dots 2^{2I_j}$   
 $j = 0, 1, \dots, n-1; T_0 = 1$

Figure 5. Tabulation of Results in This Section.

however, considerably larger than  $R_n$ , the total number of functions realizable by such a net. It follows, then, that at least one of these functions will be realized by more than one combination of dial positions. (In Reference 4 it is shown that each function can be realized in at least two ways.)

A question of interest is: How many different combinations of dial settings will realize the same function in a LADICAN? It turns out that if we classify the  $R_n$  functions according to the number of ways each is realizable, there are  $n$  different classes. That is, we can partition the set of realizable functions into  $n$  subsets, each subset characterized by the number of ways each of its member functions is realizable. This number will be designated by  $P_k$ , where  $P$  is the number of possible realizations, and the subscript  $k$  designates the subset,  $k = 0, 1, \dots, n-1$ . The following have been developed: (1) a way of determining the subset to which a (realizable) function belongs; (2)  $P_k$  for each subset; and (3) the cardinality of each subset.

The first result is not presented here because it requires the use of the partition matrix, an important tool in the A-C theory but not discussed here (see Reference 4).

Results two and three are displayed in the table of Fig. 5.

We note that subset 0 corresponds to the least number of realizations, while subset  $(n - 1)$  corresponds to the most. Further, subset 0 is the largest subset, while subset  $(n - 1)$  is the smallest. In each case, the distribution between the two extremes is monotonic. These facts are illustrated in Fig. 6.

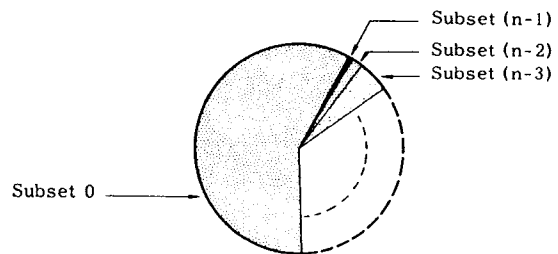


Figure 6. Representation of Subset Cardinality.

As a numerical example, let us consider the case where  $n = 3$  and each universal element has two external inputs. The table then becomes:

Subset	Number of Realizations	Cardinality of Subset
0	4	201,600
1	92	1,680
2	5,476	16

This table displays numerically some of the facts pointed out above. We see that  $(5476)(16) + (92)(1680) + (4)(201,600) = 1,048,576$ . The total number of possible dial setting combinations is

$$2_1^1 2_2^{2^1} 2_3^{2^1} = (16)(256)(256) = 1,048,576.$$

The two numbers are seen to be identical. Further, by applying Equation (4), we find that  $R_3 = 203,296$ . This agrees with the sum of the numbers in the right-hand column of the table.

Recapitulating, A LADICAN of  $n$  universal elements can realize  $R_n$  functions. These  $R_n$  functions can be partitioned into  $n$  subsets. The partition is determined by inspecting the  $A_k C_k$  partition matrix representations (see Reference 4) of each function,  $k = n-1, n-2, \dots, 1, 0$ . If the class to which a function belongs is known, then the number of possible realizations that exist for this function by this net and exactly what these realizations are can be established. Further, by knowing that the function is in class  $k$ , then we can disregard the top  $k$  universal elements as superfluous.

To evaluate a particular training algorithm on a LADICAN, it is necessary to be able to specify (to the teacher) a function that is realizable by the pupil. Such a specification may be made straightforwardly by giving a 1 or 0 specification for each of the registers in the net. However, a function specified arbitrarily in this fashion may be in any of the  $n$  subsets. For this given task, each subset may be thought of as representing a certain quality of testing function. (The word *quality* is used because functions from different subsets are dependent upon the precise settings of a different number of elements, hence, switches.) It turns out that for a function to represent the most rigorous kind of test for a training algorithm, the function must be a member of subset 0, because it is only for these functions that the precise setting of all  $n$  elements is required. Further, these functions have the least number of possible realizations, thus again reflecting the more stringent performance requirements of learning such

function. The register settings required to realize functions of this subset 0 can be made unique by arbitrarily making an a priori specification of one register in each of the first  $(n - 1)$  of the  $n$  elements in the LADICAN. This observation is based upon some facts developed in Reference 4.

Since functions of all other subsets do not require the specification of all registers in the LADICAN, the following more restricted observation can be made: In a LADICAN, the settings of those registers which specify a given realizable function can be made unique by an a priori setting to zero of a specified register in each of the first  $(n - 1)$  of the  $n$  elements. This observation will be used in developing the training algorithms. Specifically, a register designated  $V_{00}$  in each of the first  $(n-1)$  elements will be set to zero.

### 2.5 Permutation of Inputs

The number of functions realizable by a LADICAN,  $R_n$ , has been determined. Another related question is: What effect does changing the assignment of input variables (labeling) have on the set of functions realizable by such a network? Obviously, nothing happens to the cardinality of this set, for in deriving  $R_n$  there were no requirements on the labeling other than that it be fixed. Therefore, the sets of realizable functions that correspond to two different labelings of the same network can differ one from the other only to the extent of containing different functions; their cardinality must, however, be the same.

It would be of interest to determine the number of functions realizable by a LADICAN when all possible labelings are considered. This number is the cardinality of the union of the sets of functions realizable by the LADICAN corresponding to all possible labelings. Although such a result is not yet available for the general case, two general ideas that are useful in this endeavor are presented, together with a development of the result for the special case of two  $U_2$ 's connected in cascade.

The first of these two ideas is expressed in General Statement I:

- I. In any network of universal logic elements, it is clear that an interchange of inputs to an individual element will have no effect upon the set of functions realizable by the net but may change the dial settings required for realizing these functions.

With this statement in mind, consider the following specialization. Take an arbitrary network of univer-

sal elements. Let this network have  $m$  inputs, and let  $k$  be the number of elements in the net whose inputs include at least one of the  $m$  (external) inputs. Number these elements  $1, 2, \dots, k$ ; let  $i_j, j = 1, 2, \dots, k$  be the number of external inputs to the  $j^{\text{th}}$  element, and classify the external inputs according to the element they feed. Assume that this classification is disjunctive. We then have an "ordered" partitioning of the inputs into  $k$  classes; the cardinality of each class is  $i_j, j = 1, 2, \dots, k$ , and

$$\sum_{j=1}^k i_j = m.$$

A simple calculation shows that there are

$$\frac{m!}{\prod_{j=1}^k (i_j!)}$$

ways of making such an assignment of the  $m$  inputs into  $k$  disjoint classes of cardinality  $i_j, j = 1, 2, \dots, k$ . It follows for this case that of the  $m!$  possible permutations of  $m$  inputs, this number

$$\frac{m!}{\prod_{j=1}^k (i_j!)}$$

is the number of permutations not excluded by Statement I as candidates for affecting the set of functions realizable by the given net. These considerations lead to the second idea, General Statement II:

II. In any network of universal elements, if the  $m$  input variables are partitioned as indicated above, then only the

$$\frac{m!}{\prod_{j=1}^k (i_j!)}$$

permutations corresponding to repartitioning (in the ordered sense) the variables can affect the set of functions realizable by this net.

Consider the LADICAN of the two  $U_2$ 's shown in Figure 7. We know from General Statement I that interchanging the variables  $x_1$  and  $x_2$  or  $\phi$  and  $x_3$  will not change the set of functions realizable by this network. And from General Statement II we know that only the  $3!/(2!)(1!) = 3$  permutations corresponding to the (appropriate) repartitions of the variables can affect the set of functions realizable by this network. These three partitions are: (a)  $x_1x_2|x_3$

(in Figure 7), (b)  $x_1x_3|x_2$ , (c)  $x_2x_3|x_1$ . For reasons indicated in Reference 4, we will call (a) the  $x_3$  viewpoint (b) the  $x_2$  viewpoint, and (c) the  $x_1$  viewpoint. To each partition (viewpoint) there corresponds a set of  ${}_2R_2 = 88$  functions that can be realized. The cardinality of the union and intersections of these three sets of realizable functions is of interest. The possibilities are indicated in Figure 8.

In Reference 4, it is shown that the shaded intersections of Figure 8 are empty; that is, a function is realizable either from only one viewpoint, or from all three viewpoints—there are no functions realizable from only two viewpoints. Figure 9 represents these facts.

It turns out that the cardinality of the intersection is 56, and the cardinality of each of the "wings" is 32. Thus, the cardinality of the union is 152.

### 2.6 Training Algorithms

In this subsection, three algorithms for training LADICANs are presented. The fundamental one is Algorithm 1; Algorithms 2 and 3 use it as a building block. In Reference 4, it is proved that

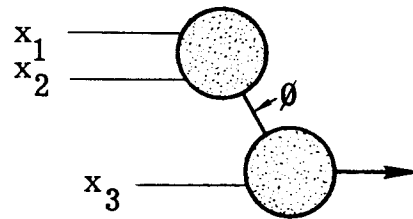


Figure 7. LADICAN of Two  $U_2$ 's.

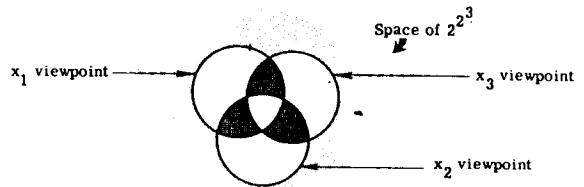


Figure 8. Possible Intersections of Three Viewpoints.

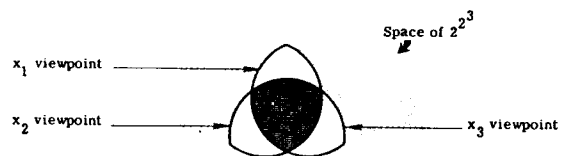


Figure 9. Actual Intersections of Three Viewpoints.

when Algorithm 1 is applied to a two-element LADICAN, it will assure convergence to a solution if one exists.\* Algorithms 2 and 3 are for training LADICANs of  $n$  universal elements,  $n > 2$ . Algorithm 2 is used in situations where the input environment is not constrained; Algorithm 3 may be applied to constrained environments as well. The important feature of these last two algorithms is their ability to identify and preserve good information once it is obtained.

### 2.6.1 Algorithm 1

Consider a universal element with  $n$  inputs; it contains  $2^n$  registers, and each of these registers can count from 0 to 1 in  $s$  equal steps (of  $1/s$ ). When the register is at 0 or 1, we say that it is set; otherwise, it is in transition. As discussed earlier, the zero corresponds to the register's operation as an open switch, and the 1 corresponds to its operation as a closed switch. In each of its positions, the register will have a probability,  $p_i$ , of acting as though there were a 1 in it, and a probability,  $q_i = 1 - p_i$ , of acting as though there were a 0 in it. The subscript  $i$  corresponds to the count  $i/s$  in the register,  $i = 0, 1, \dots, s$ . Thus  $p_0 = 0$  and  $p_s = 1$ .†

Consider a LADICAN of two universal elements as shown in Figure 10.

Element 1 has  $I_1$  registers; element 2 has  $2I_2$  registers (the multiplier 2 is due to the extra input  $\phi$ ). Arrange the  $2I_2$  registers of element 2 into columns,  $V_0$  and  $V_1$ , each of  $I_2$  registers. Each row of these two columns (placed side to side) corresponds to one of the  $I_2$  minterms of the  $i_2$  inputs; the appropriate column is selected by  $\phi$ . By a similar construction, element 1 has one column,  $V_0$ , of  $I_1$  registers.

We are interested in setting the registers of these elements to perform a particular one of the switching functions realizable by this net. To make the required setting unique, we set one register of column  $V_0$  of element 1 to zero and do not change it from that value. This done, there corresponds only one correct setting at 0 or 1 for each of those registers that is required to specify the given function. This uniqueness is required in the convergence proof that is to be outlined.

\* To the authors' knowledge, this is the first algorithm which assures convergence to a solution (if it exists) when more than one interacting adjustable logic element is included in the training.

† This model of the register is similar to the statistical switch on the SOBLIN of Reference 8.

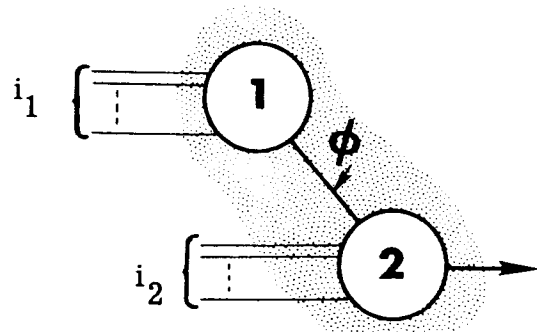


Figure 10. LADICAN of Two Universal Elements.

Under certain conditions, some registers are not called upon to specify the given function; therefore, they do not have required settings and will be said to be set correctly regardless of their count at any given time.

With reference to a specified function, then, each register may be set correctly, incorrectly, or be in transition (a transition state is one wherein  $p_i \neq 1$  or 0), and the letters  $C$ ,  $I$ , and  $T$ , respectively, will so indicate. We denote the total number of registers by  $M = I_1 + 2I_2$ . Since each register has  $(k + 1)$  possible levels, there are  $(k + 1)^M$  possible patterns of register settings. For each of these  $(k + 1)^M$  patterns, depending upon the specified function, we will assign a  $C$ ,  $I$ , or  $T$  to each register. We will call each of these resulting patterns of  $C$ ,  $I$ , and  $T$  a point in which we now define as the *register setting space*.

In the convergence proof of Algorithm 1, the task will be to show that a training program based upon this algorithm will yield the desired register settings in a two-element LADICAN. In terms of the register setting space, this may be stated as follows: beginning at some arbitrary point in the space, the algorithm must take us, via some path, to the point representing "all registers set correctly." Call this point  $F$ .

For that proof, it is convenient to map the register setting space into the real line (specifically, the non-negative integers). Denote the image of  $F$  by  $F^*$ . This transformation is made one-to-one at point  $F$ ; i.e., the inverse image of  $F^*$  is uniquely  $F$ . The Euclidean metric is employed in this image space as a tool to establish that the algorithm takes us to the point  $F^*$ . Then, since the inverse image of  $F^*$  is uniquely  $F$ , we assert that the algorithm takes us to  $F$  in the register setting space.

We refer to each point in the register setting space by  $r_j$ ,  $1 \leq j \leq 3M$ . To each point  $r_j$  there corresponds a number of correct register settings

– call it  $A_j$  – and a number of incorrect register settings – call it  $B_j$ .

$$A_j = \Sigma \text{ correct register settings at } r_j$$

$$B_j = \Sigma \text{ incorrect register settings at } r_j$$

Define the mapping  $\sigma$  as follows:

$$\sigma : r_j \rightarrow M - (A_j - B_j)$$

At the point  $F$ ,  $A = M$  and  $B = 0$ ; therefore  $F^* = 0$ . (No other point is mapped onto 0). There are no points mapped into the negative real line, and the largest positive integer that is mapped onto is  $2M$  (all registers set incorrectly). We call the points on the real line  $y_i$ ,  $i = 0, 1, \dots, 2M$ . Except for  $y_0 = F^* = 0$ , and  $y_{2M} = 2M$ , each  $y_i$  corresponds to a set of  $r_j$ 's because this mapping is many-to-one.

We use the Euclidean metric,

$$\rho(y_i, y_j) = |y_i - y_j| = |r_i\sigma - r_j\sigma|, \begin{matrix} r_i \in y_i\sigma^{-1} \\ r_j \in y_j\sigma^{-1} \end{matrix}$$

We specialize this to measure the distance from the origin:

$$\rho_0(y_i) = \rho(y_i, 0) = |y_i - 0| = |r_i\sigma - 0| = \begin{matrix} r_i\sigma, r_i \in y_i\sigma^{-1} \end{matrix}$$

This metric  $\rho_0$  specifies the distance from a given set of register settings  $r_i$  to the desired solution. Thus, at the solution point,

$$\rho_0(F^*) = 0$$

*Now Algorithm 1:* Start with all registers set at  $1/2$  (if  $s$  is odd, pick a level near  $1/2$ ), except one register of column  $V_0$  of element 1 which is set to 0. Specify a function. Apply an input. If the output of the net is correct, apply another input. If the output of the net is incorrect, ascertain which register determined the output of each element and “punish” each of them. That is, if a register acted like a 1, decrement it by  $m/s$ ; if a register acted like a 0, increment it by  $m/s$ , where  $m$  is an integer between 0 and  $s$  specified a priori. (For notational convenience  $m/s$  will hereafter be designated by  $\Delta$ .) Apply another input and continue. If a register is set at 1, an increment leaves it unchanged; similarly, if it is set at 0, a decrement leaves it unchanged. The requirement on the input sequence is that the frequency of occurrence of the minterms to each element must be essentially evenly distributed. A weaker condition would also suffice here. Namely, each minterm to each element must appear at

least “every once in a while,” that is, infinitely often in an infinite sequence. These requirements on the element minterms are considerably weaker than a similar requirement on the entire network minterms would be.

At each iteration, the inputs applied to the net will select one register in each element. Each register could have been set  $C$ ,  $I$ , or  $T$ . If the output of the net is correct, no action is taken; therefore, the register settings do not change. However, if the output of the net is incorrect, a “punish” action is taken and, consequently, the register settings will be changed. We denote the settings of the two registers selected in each case by an ordered pair of letters, the first letter corresponding to the register of element 1, and the second to the register of element 2. The possible settings of the two registers at the beginning of any cycle are  $II$ ,  $IT$ ,  $IC$ ,  $TI$ ,  $TT$ ,  $TC$ ,  $CI$ ,  $CT$ , and  $CC$ .

In Reference 4, a state diagram is constructed to represent all possible starting settings of the two registers and all possible transitions from these settings resulting from a “punish” action. The change in the value of the metric  $\rho_0$  in going from one state to the other is computed for each state. This information is then used to prove that for  $\Delta \geq 1/2$ ,  $\rho_{0s}$  goes to zero in a finite number of iterations, where  $\rho_{0s}$  is the distance of the register settings in the pupil to the correct solution. Thus it follows that Algorithm 1, with  $\Delta \geq 1/2$ , guarantees attainment of the correct register settings by a two-element LADICAN in a finite number of steps.

Two modifications of Algorithm 1 are considered in Reference 4. In the first, instead of performing a “punish” action after an incorrect output, perform a “reward” action after a correct output. That is, if the output of the net is correct, ascertain which register determined the output of each element and “reward” each of them. That is, if a register acted as though it contained a 1, increment it by  $\Delta$ ; if a register acted as though it contained a 0, decrement it by  $\Delta$ . In the second modification of Algorithm 1, each time the net gives a correct answer, perform a “reward” action, and each time the net gives an incorrect answer, perform a “punish” action. Algorithm 1 is then referred to as the “punish-only” method; the first modification, “reward-only;” and the second modification, “reward-punish.”

It is shown that the “punish-only” method guarantees a solution, but that the “reward-only” method does not guarantee a solution. It is then argued, and verified experimentally (see Figure 15), that if the “reward-punish” method does yield a

solution, it does so in a considerably shorter time than the "punish-only" method. However, there are cases where the "reward-punish" scheme does not yield a solution (in a reasonable amount of time.)

Another experimental fact is that even when none of the registers is fixed a priori (so that the solution is not unique), the "reward-punish" algorithm still yields correct solutions (whereas the "punish-only" algorithm does not converge). In fact, as shown in Figure 15, the best operation is obtained using the "reward-punish" algorithm in this way. Because the solution is not unique when a register is not fixed a priori, it is more difficult to prove that this method will assure a correct solution. Such a proof is not yet at hand.

### 2.6.2 Algorithm 2.

One of the major problems in training a many-element net is to determine which elements in the net are wrong when the pupil gives the wrong output. The reason for this difficulty is that there is no a priori knowledge of what function each element in the net is to perform to make the net's performance correct.

Algorithm 2 solves this problem by performing a masking operation on the inputs to the LADICAN. Either all inputs to a given element are masked, or none of them is. When the inputs to an element are masked, the element receives all zero inputs. This masking operation allows us to present selected subsets of the external inputs to the pupil at a given time. In so doing, the training activity can be restricted to a small segment of the net. Algorithm 1 is used to train this small segment. Then the correctly set registers of this segment are identified and not changed thereafter. The entire net is covered in this piecemeal fashion. This approach avoids the difficulty of not being able to determine which element is at fault when an error is made by the net.

This algorithm is confined to the case where the input set is unconstrained. This is because the pupil "sees" the input only after the masking operation has been performed. This modified input must still be a legitimate input—i.e., a member of the teacher's set of care terms. This can be guaranteed in general only if the set of care terms is the entire input set.

A network of 169 elements with 507 external inputs was successfully trained using this algorithm. The function on which it was trained was a member of class 0.

The details of this algorithm are reported in Reference 4.

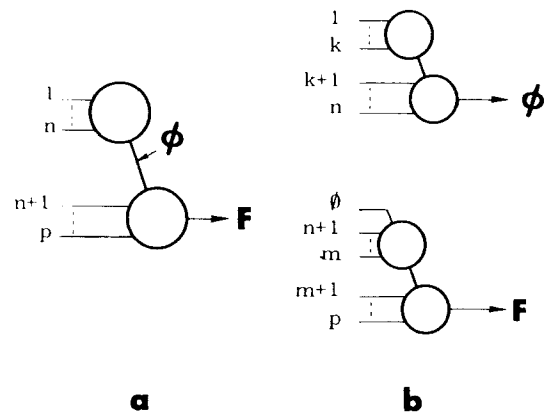


Figure 11. Basic Step in Algorithm 3.

### 2.6.3. Algorithm 3

Algorithm 3 solves the following problem.\* Given  $m$  inputs, find the LADICAN of maximum length whose performance space contains a specified (by the teacher) function of the  $m$  inputs.

The performance space of a one-element LADICAN with  $m$  inputs will certainly contain the solution, for such a LADICAN is universal. Therefore, our first try will be with a two-element LADICAN: connect the inputs to an  $m$ -input, 2-element LADICAN (see Fig. 11a), and train the LADICAN with Algorithm 1 until a solution is reached (if it exists). After the training is completed, sequence through all the care terms† and record the output  $\phi$  for each. We now have available the functions to be performed by elements 1 and 2 for correct network performance.

Next, replace element 1 with another two-element LADICAN (see Fig. 11b), with sufficient inputs to accommodate the inputs that went to element 1. Train this two-element LADICAN (using the previously recorded information as the desired function) until it too, reaches a solution, if one exists. Similarly, replace element 2 with a two-element LADICAN with enough inputs to accommodate the inputs that went to element 2, plus one input to accommodate the output  $\phi$  of element 1. Train this two-element LADICAN, using the recorded information to provide the input  $\phi$ , until it reaches a solution (if one exists).

\* The motivation for this problem will be described in a forthcoming paper.

† For this reason, Algorithm 3 is more readily applied with a constrained input set.

Continue to break down the elements, two at a time, until the maximum-length LADICAN whose performance space contains the solution is found. Since this algorithm performs an iterative structural decomposition (by making a change in structure to accomplish its desired aim), it is a first step towards realizing the restructuring mode of the general algorithm discussed in the Introduction. We note that, by construction, this algorithm also guarantees attainment of a solution if one exists.

Although the proof of convergence of Algorithm 1 was made only for a two-element LADICAN, Algorithm 1 can be applied to many-element LADICANs.\* At present, however, we do not have a proof of convergence for more than two elements. Experimentally, the authors have never experienced difficulty with applying Algorithm 1 directly to LADICANs of three or four elements. Difficulty was experienced in applying Algorithm 1 directly to LADICANs of five or more elements; this was, however, overcome by using Algorithm 2 or 3.

These experimental facts are important to Algorithm 3 for two reasons: (1) For a given set of  $m$  inputs, if we start with a four-element LADICAN, the size of universal element required is about half that required for a two-element LADICAN; (2) With universal elements of the maximum available size,  $m$  can be twice as big with a four-element LADICAN.

### 2.7 Experimental Results

This subsection presents experimental results pertaining to LADICANs of universal elements. These results were obtained by simulating the networks on a digital computer.

The pupil subroutine portion of the simulation program is capable of simulating several particular realizations of the general model of the register discussed in Section 2.6.1. We recall that the counters can be incremented or decremented in steps of  $\Delta = m/s$ . In this simulation,  $s = 63$ ; hence, there are 64 possible states. In one of the realizations, the  $p_i$  assignments are a linear function of the register count; namely,  $p_i = i$ , where  $i$  is the count in the register (Fig. 12). We refer to this realization as the *random-firing* method. In a second realization, the  $p_i$  assignments are nonlinear; namely,  $p_i = 0$  if  $i \leq \theta$ , and  $p_i = 1$  if  $i > \theta$  (Fig. 13). We refer to this realization as the *fixed-firing* method.

Except where noted,  $\Delta$  is held constant in the investigations reported.

\* To make the solution unique in such a net, one register in each element except the bottom one is set and held at 1 or 0.

#### 2.7.1 Early Experiments

The earliest experiments were concerned with the "learning dynamics" of a LADICAN of two  $U_2$ 's. The training algorithm used was essentially Algorithm 1 with "reward-punish." The only difference was that no registers were fixed a priori.

As discussed earlier, this network with a fixed labeling is capable of realizing 88 of the 256 possible functions of three inputs. The first question was: Is this algorithm capable of getting the pupil to achieve each of the 88 realizable functions? If so, are there any discernible characteristics of the learning dynamics peculiar to those functions it can learn? The answer to both of these questions turned out to be in the affirmative.

Extensive experiments with this network revealed the following:

- (1) The achievability of realizable functions does not depend upon  $\Delta$ , except for  $\Delta = 1$ . Tests were run with  $\Delta = 1/63$  to  $\Delta = 1$  in steps of  $1/63$ , and the only one for which training failed was  $\Delta = 1$ . This is to be contrasted with later experiments in which the solution was made unique by fixing a register. In this case ( $\Delta = 1$ , punish only, random or fixed-firing schemes), solutions were attained as predicted by theoretical consideration.

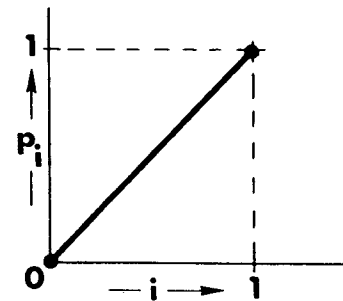


Figure 12. Register Realization 1: Random-Firing Method.

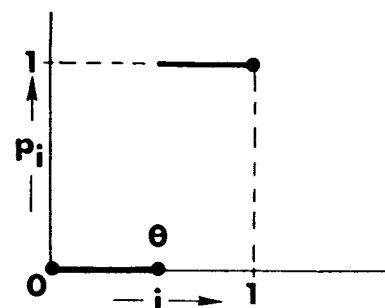


Figure 13. Register Realization 2: Fixed-Firing Method.

- (2) Since no register was fixed a priori, the solution was not unique, and it turned out that the terminal register settings changed as a function of  $\Delta$ . These represented alternate equivalent realizations of the given function.
- (3) For  $\Delta = 1/63$ , all 88 realizable functions were achieved when the input sequences were arbitrary. However, when the same input sequence was used to train the net for each of the 88 realizable functions, in seven cases the

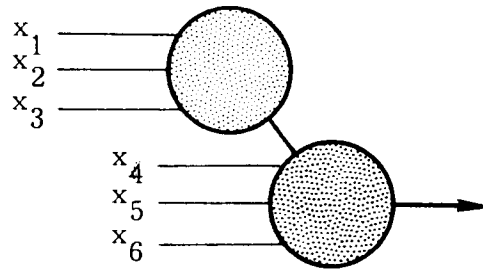


Figure 14. LADICAN Used in Algorithm 1 Experiments.

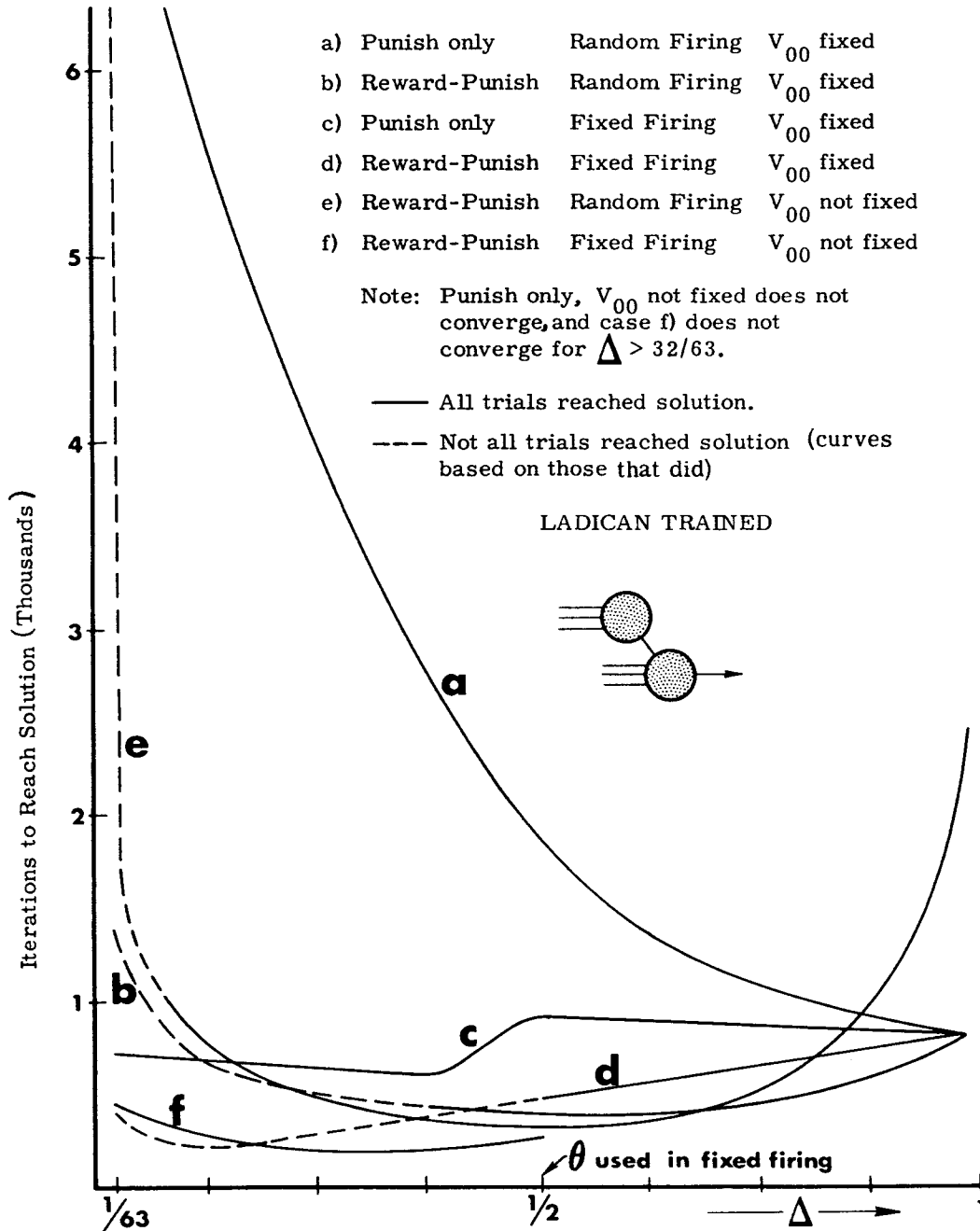


Figure 15. General Characteristics of Mean-Iteration-Count-To-Reach-Solution vs.  $\Delta$  Using Algorithm 1.

pupil became "trapped" in some erroneous states that it was unable to leave (for trials of up to 8,000 iterations). The other 81 functions were readily achieved.

- (4) Fixed firing schemes worked well with all  $\theta$ 's tried. These trails ranged from  $\theta = 0$  to  $\theta = 48/63$ .
- (5) When the network attempted to learn a non-realizable function, each register acted in an unpredictable fashion, and one or more of them would not settle down to an extreme value. In the dynamics of the individual registers there were no apparent characteristics upon which to base a criterion for distinguishing between realizable and nonrealizable functions.
- (6) The dynamics of the network as a whole displayed some promising characteristics for developing such a criterion, and one that was formulated on the basis of these characteristics is given in Reference 4.

2.7.2 Experiments with Algorithm 1

Some extensive investigations were carried out on the two-element LADICAN of Figure 14 using variants of Algorithm 1. Various combinations of "punish-only," "reward-punish," random firing, fixed firing,  $V_{00}$  fixed,  $V_{00}$  not fixed were considered. For each of these cases, the number of iterations required to achieve realizable functions was tabulated as a functions of  $\Delta$ . These data were averaged and smoothed, and the resulting general characteristics of the mean-iteration-count-to-reach-a-solution vs.  $\Delta$  are displayed in Figure 15.

2.7.3 Experiments with Algorithm 2

The simulation program was not capable of simulating Algorithm 2 in its entirety. Certain specializations were made;<sup>4</sup> Figure 16 shows the

results of experiments using this specialized version of Algorithm 2 on LADICAN networks of universal elements of three external inputs each.

2.7.4 Experiments with Algorithm 3

The experimental results with Algorithm 3 will appear in a forthcoming paper.

CONCLUSION

By using the Ashenurst-Curtis theory of decomposition of switching functions, the authors have done the following:

- 1) Determined  $R_n$ , the cardinality of the set of functions realizable by a LADICAN of  $n$  universal elements
- 2) Classified the  $R_n$  functions by the number of ways each is realizable by a LADICAN
- 3) Developed a way to determine to which resulting partition each of the  $R_n$  functions belongs
- 4) Determined the cardinality of the set of functions realizable by a LADICAN of two  $U_2$ 's when all possible labelings are included
- 5) Developed an algorithm for training two-element LADICANS and its attendant proof of convergence
- 6) Developed two algorithms for  $n$ -element LADICANS. These algorithms have the important ability to identify and preserve good information once it is acquired by the pupil.

Much work remains to be done in developing the self-organizing machine described in the introduction of Section II. The questions pertaining to the cascade-type structure that have been left unanswered will have to be resolved. Furthermore, properties of different kinds of structures will have to be determined, particularly properties of a

No. of elements	5	10	15	20	34	169*
No. of external inputs	15	30	45	60	102	507
No. of trials	10	10	10	5	3	1
Mean iteration count to reach solution	1,592	3,781	5,645	7,560	13,530	79,968
Mean iteration count per element	318	378	377	378	398	473

\*The 169 element LADICAN represents the maximum capacity of the simulator program.

Figure 16. Tabulation of Data From Experiments Using the Specialized Version of Algorithm 2 on LADICANS of Universal Elements with Three External Inputs Each.

parallel-type structure and of a hybrid parallel-cascade-type structure. Even with these results, a nondisjunctive assignment of the inputs will have to be considered. (The A-C theory is general enough to handle all these cases.)

It appears that all this information will be a necessary prerequisite for the one big achievement that will make the realization of the aforementioned self-organizing machine possible—the determination of an algorithm for making meaningful changes in the structure of the pupil network during training to assure that the solution is contained within the pupil's performance space. The ability to do this will, of course, depend upon the development of a criterion for deciding when to make these changes.

#### REFERENCES

1. ASHENHURST, R. L., "The Decomposition of Switching Functions," *Proceedings of an International Symposium on the Theory of Switching*, Harvard University Press, 1959
2. CURTIS, H. A., *A New Approach to the Design of Switching Circuits*, D. van Nostrand Company, Inc., New York, 1962
3. LENDARIS, G. G., "On the Definition of Self-Organizing Systems," *Proceedings, IEEE*, (correspondence) Vol 52, Mar 1964, pp 324-325
4. LENDARIS, G. G., and STANLEY, G. L., "On the Structure-Dependent Properties of Adaptive Logic Networks," GM Defense Research Laboratories Technical Report, TR63-219, Jul 1963 (This report is available by writing the authors at GM DRL, Box T, Santa Barbara, California).
5. Singer, T., "The Decomposition Chart as a Theoretical Aid," Harvard Computation Lab. Report, No. BL-4, Sec. III, 1953
6. Maitra, K. K., "Cascaded Switching Networks of Two-Input Flexible Cells," *IRE Transactions on Electronic Computers*, Apr 1962
7. HAWKINS, J. K., "Self-Organizing Systems—A Review and Commentary," *Proceedings, IRE*, Vol. 49, Jan 1961, pp. 31-48
8. Carne, E. B., "A Study of Generalized Machine Learning," Technical Documentary Report, ASD-TDR-62-166
9. SHANNON, C. E., and MCCARTHY, J., *Automata Studies*, Princeton University Press, 1956
10. ROSENBLATT, F., "The Perception—A Theory of Statistical Separability in Cognitive Systems," Cornell Aeronautical Lab. Report, No. VG-1196-G-1, Jan 1958
11. ROSENBLATT, F., "Perceptron Simulation Experiments," Cornell Aeronautical Lab. Report, No. VG-1196-G-3, Jun 1959
12. WIDROW, B., "Pattern Recognition and Adaptive Control" *Proceedings of AIEE Symposium on Discrete Adaptive Processes*, JACC, Jun 1962
13. HAWKINS, J. K., MUNSEY, C. J., and STAFFORD, R. A., "Research on Biax Type Elements and Associated Circuits (Biax Perceptron)" Annual Summary Report, No. V-2111, Ford Motor Company, Aeronutronic Division, Jan 1963
14. LENDARIS, G. G., and STANLEY, G. L., "An Opticalogical Self-Organizing Recognition System," *Proceedings of the Symposium on Optical and Electro-Optical Information Processing Systems*, M. I. T. Press, 1965.