THE UNIVERSITY OF MICHIGAN
COLLEGE OF LITERATURE, SCIENCE, AND THE ARTS
Communication Sciences Program

Quarterly Report No. 2

15 July 1962 - 15 October 1962

MACHINE ADAPTIVE SYSTEMS

Arthur W. Burks
J. Willison Crichton
Marion R. Finley

ORA Project 05089

under contract with:

DEPARTMENT OF THE ARMY
U. S. ARMY SIGNAL SUPPLY AGENCY
CONTRACT NO. DA-36-039-SC-89085
FORT MONMOUTH, NEW JERSEY

administered through:

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

October 1962

TABLE OF CONTENTS

TOWARD A THEORY OF AUTOMATA BASED ON MORE REALISTIC PRIMITIVE ELEMENTS

Arthur W. Burks

LIST OF FIGURES

1. Introduction*

We will begin with automata composed of the idealized primitive elements usually studied in automata theory, and then consider automata based on primitive elements which simulate more closely the actual operation of both artificial and natural systems. We are particularly interested in comparing the computational powers and rates of these more "realistic" automata with the computational powers of idealized automata.

2. Idealized Automata

An idealized automaton is constructed from instantaneous "nan" (not-and) switches and unit delays whose initial output states are either one or zero (Fig. 1). Unlimited branching ("fan-out") is allowed, but all cycles must pass through delays. These primitive elements operate synchronously, with time having the integral values 0, 1, 2, ··· .

A finite idealized automaton is composed of a finite number of elements. Those wires which are not driven by other wires are called the inputs of the automaton; certain wires are designated as the outputs of the automaton.

There does not exist a single satisfactory general definition of an infinite automaton, and we will consider only one special case. A Turing machine

---

*The writing of this paper was supported by the U. S. Army Signal Corps through Contract DA-36-039-SC-89085 and Grant DA-SIG-36-039-61-G4.

---

consists of a finite idealized control automaton attached to an infinite tape. The tape itself is an idealized automaton. It is an infinite shift register, that is, an infinite iterative array of finite idealized automata. Each finite automaton of this linear array is the same except that a finite number of them may have delays whose initial outputs are one instead of delays whose initial outputs are zero.

Our only purpose in having both kinds of unit delays is to be able to vary the kind of delays in an automaton (i.e., the initial state) while leaving the rest of the automaton unchanged, as in the present instance. Either kind of delay can be synthesized from the other by means of the nan switch.

This is the same as saying that only a finite number of squares of a tape may be marked initially; all the rest must be blank. Since the control automaton is finite, only a finite amount of information can be stored in a Turing machine initially. It follows that at each time t only a finite amount of information will be stored in a Turing machine, though this quantity of information can grow without bounds. Note that a Turing machine in our sense has a finite number of input wires and a finite number of output wires, in contrast to Alan Turing's original conception [1].

Since actual switching takes time, this separation of primitives into instantaneous switches and pure delays is clearly an idealization. It is a useful idealization for two reasons. First, Boolean algebra (i.e., truth-function theory) may be applied simply and directly to the analysis of instantaneous switches, whereas the analysis of switches with built-in delays is more complicated. Second, the time actually required for switching varies with the

state of technology, and hence it is desirable to have both instantaneous switching primitives and switching elements with built-in delays, the former for studying what is needed from a purely logical point of view, the latter for studying what is needed from a technological point of view.

3. Behavior and Computation

Let there be given a finite alphabet of input {output} characters or states. Let I be the set of all infinite sequences of type omega (i.e., corresponding to the numbers 0, 1, 2, ...) of input characters, and let Θ be the set of all infinite and finite sequences of output characters.

A finite sequence may occur as the computed output of an automaton; see below.

When a sequence $< i_0, i_1, i_2, \ldots >$ of I is fed into the inputs of an automaton at times 0, 1, 2, ... respectively the automaton will produce an output sequence $< \theta_0, \theta_1, \theta_2, \ldots >$ belonging to Θ; the <u>behavior</u> of an automaton is the mapping it produces from I into Θ. Finite automaton behaviors have been studied intensively (e.g., [2], [3]), but though the behaviors of Turing machines are of interest ([4], [5], [6]) the concept of behavior is not nearly so important for the theory of Turing machines as the concept of "computation." This latter concept has been defined in a number of different ways, none of which is applicable to finite automata, so we will offer a new definition of computation.

This is an extension of the definitions of [4], p. 285.

Each sequence of I{Θ} may be regarded as an input {output} tape. An input tape may be placed in the input tape reader R of Figs. 2 and 4, which works as follows. The tape symbol being scanned at time t appears at the output instantaneously. When the "read control" wire is activated (put in state one) at time t the tape reader advances the tape one square, so that at time t+1 the next symbol appears on the output of the tape reader and hence at the input of the automaton. If the read control is inactive at time t the tape is not advanced, so that in this case the symbol which appeared at the output of the tape reader at time t also appears there at time t+1. The output tape writer W of Figs. 3 and 4 works similarly: whenever the "write control" wire is activated the tape symbol being presented to the tape writer by the automaton is recorded on the output tape and the tape is advanced to the next blank square. Note that the read and write control wires are outputs of the automaton, so that it controls their actions. Both reader and writer can only move their tapes forward, so that neither can be used for intermediate storage during a computation. The computation of an automaton is the mapping it produces from I into Θ when the sequences of I are supplied to the input tape reader and the sequences of Θ are produced by the output tape writer (see Fig. 4).

It is clear from this definition that behavior is a special case of computation: that case in which the read and write control wires are always active (in state one). That is, behavior is "real-time" computation.

The foregoing definitions of behavior and computation also apply to the slow automata and cellular automata to be introduced later.

It is of interest to compare finite idealized automata and Turing machines

with respect to certain problems concerning behavior and computation. Consider, for example, the four problems: Is there an algorithm (i.e., Turing machine) for deciding whether any two < finite idealized automata > {Turing machines} have the same (behavior) [computation]? There is an algorithm for deciding whether any two finite automata have the same behavior ([2], [3], [7]), but there is no algorithm for deciding whether any two Turing machines produce the same computation.

It can be shown that if there were there would be an algorithm for deciding whether or not an arbitrarily given Turing machine will "halt," and Turing [1] showed that there is no such algorithm. In our terminology a machine which halts is a machine which records only a finite number of output characters on the output tape.

The solutions to the other two problems are unknown.

Friedman [8] gives a solution for the case of finite automata in which the read control wire is always active (i.e., the input tape advances one square every unit of time). See also Burks and Wright [9].

---

4. Slow Automata

The automata we consider next are intended to reflect, to some degree, the physical facts that time is required for switching and that the number of wires a single component can drive is limited by power and time requirements. The primitives we will use are shown in Fig. 5. Every "nan" switch is added fol-

lowed by n units of delay. An "and" switch is added which has a units of delay. The output wire of an element can be connected to at most one input wire. Hence when information is to be sent to two or more places a branching or fan-out element must be used; this element has b units of delay.

The output of the (nan) {and} [branching] element is inactive for the first (n) {a} [b] units of time.

It is stipulated that an element output wire which is designated as the output of the automaton can be connected to nothing within the automaton itself, but is to be reserved to operate something else, e.g., a tape writer. Any automaton constructed from a finite number of slow elements by these rules is called a finite slow automaton.

While the slow primitives of Fig. 5 are more realistic than the idealized primitives of Fig. 1, we could have chosen an even more realistic set of primitives. For example, we might have chosen four switching primitives realizing pq, $p\overline{q}$, $\overline{p}q$, and $\overline{pq}$, each followed by a unit delay, and a branching element with four outputs and a unit delay. The reason for our less realistic choice is as follows. When one is interested in a particular type of equipment he should choose a set of slow primitives which mirrors the properties of this equipment. But we are interested in the general computational powers of automata composed of slow elements, and have chosen the primitives of Fig. 5 because they can be made arbitrarily slow by choosing n, a, and b to be large. It will be seen that the results we give below are easily extended to cover faster sets of primitives.

We will now compare finite slow automata with finite idealized automata. It is clear that corresponding to each slow automaton there is a behaviorally equivalent idealized automaton. The converse is not the case, however. Given a finite idealized automaton, in general there is no slow automaton behaviorally equivalent to it, because the switching done instantaneously in the idealized automaton takes time in the slow automaton. But for each idealized automaton one can construct a computationally equivalent slow automaton. More particularly, there is an algorithm which, when supplied with the design of a finite idealized automaton A_i, will design a finite slow automaton A_s computationally equal to A_i. We will call this our first algorithm. There is not time to present this algorithm in detail, but we will sketch the essential parts of it.

It contains a subalgorithm which operates on an idealized finite switch S to produce a slow switch S' which realizes the same truth-function as S but with a fixed delay d, the length of the delay depending on the switching function. S' is constructed from S as follows. Each "nan" element of S is replaced by a slow "nan" element with n units of delay. Then each branch of S is replaced by a tree of branching elements.

This must also be done where a wire of S is used both as an output wire for the whole switch and to drive another wire within the switch.

Finally sequences of delays are introduced into the lines so that the timing is correct, i.e., so that for each point of switch S' the transit times over all paths from the inputs to this point are equal, and so that the transit times through the switch from inputs to outputs are all the same integer d.

Let α and β be finite binary sequences and let Z^c be a sequence of c zeros. Our first algorithm also contains a subalgorithm which when given α,β designs a slow automaton (without inputs) whose behavioral output is the infinite sequence $Z^c\alpha\beta\beta\beta\cdots$, where the index c is a function of α and β. Each finite binary sequence α,β is formed by generating a single pulse, sending it along paths of different lengths, and merging these paths. β is repeated indefinitely by means of a cycle. Hence any ultimately periodic sequence $\alpha\beta\beta\beta\cdots$ can be produced by a slow automaton after some fixed delay c.

Our first algorithm now operates on the given idealized automaton A_i as follows. It puts A_i in the normal form of Fig. 6, where the switch S_i is instantaneous. It then replaces this switch S_i with a slow switch S_S which does the following. First, S_S realizes the same switching function as S_i but with delay e. Second, S_S receives the initial state of A_i at some appropriate time t when signaled to do so by a control pulse. Third, under the direction of an ultimately periodic control sequence the slow switch S_S only allows the tape read and tape write control wires to be active at time $t + (e+1) - 1$, $t + 2(e+1) - 1$, $t + 3(e+1) - 1$, $\cdots$. As a final step the first algorithm adds the ultimately periodic circuits needed for controlling S_S and to supply the initial state of A_i to S_S.

The result of the first algorithm is a slow automaton A_S which mimics the behavior of A_i at a slow rate. A_i starts in some internal state at time zero; A_S starts in this state at a later time. A_i makes a transition from one internal state to the next in one unit of time; A_S makes the same transition in e+1 units of time. What A_S does in between these times does not

matter, because the control signals prevent these intermediate actions from affecting the operation of the input and output tapes. Consequently, A_S produces the same computation as A_i.

Clearly the rate of computation of A_S is only a fraction $(e+1)^{-1}$ of the rate of computation of A_i. The question naturally arises: Is there a slow automaton which computes as fast as any given idealized automaton? In general there is not, because the decision as to whether or not to advance the input tape may depend on the tape symbol being scanned, and while this decision can be made instantaneously in an idealized automaton, it takes time in a slow automaton. This argument does not apply when the read control output from the idealized automaton is input-independent. In this case the read control signal is ultimately periodic and can be produced by a slow automaton (see the first algorithm). Moreover, there is an algorithm which, when given an idealized automaton A_i, will synthesize a finite slow automaton A_S* whose behavior is the same as that of A_i except that the outputs of A_S* are delayed by some fixed amount g.

This algorithm is given in [10]. According to this reference the algorithm was discovered by a number of different people.

A control circuit and gate may be added so that the write control wire cannot be activated before time g. Combining all this we get our second algorithm. When given the design of an idealized finite automaton A_i whose read control output is ultimately periodic, this second algorithm designs a computationally equivalent finite slow automaton A_S* which computes at the same rate as A_i

after an initial delay g.

The slow automaton A_S* produced by the second algorithm will generally have many more primitive elements than the slow automaton A_S produced by the first algorithm. Hence the greater speed of A_S* is obtained at the cost of additional equipment. It would be of interest to automata theory to have some general theorems governing interchanges of speed and equipment of this sort.

There is an extended concept of computation which is of interest. Let the <input tape reader scan> [output tape writer print] up to h squares at once and allow it to be advanced from 1 to h squares in a time step. Then for each idealized finite automata A and each integer h there is a finite idealized automaton A'' which is computationally equivalent to A but computes h times as fast.

We can consider Turing machines only briefly. A Turing machine consists of a finite control automaton connected to an infinite shift register, and in our original definition these were both constructed from idealized primitive elements. Both the finite control automaton and the infinite shift register can also be constructed from slow primitive elements so that for each Turing machine made of idealized primitives there is a computationally equivalent Turing machine made of slow primitives. There is not time to give the construction here.

5. Probabilistic Automata

The automata we have discussed so far are all deterministic. Actual computers make errors, and natural systems often function in non-deterministic, statistical ways. Probabilistic automata are intended as models which mimic, to a first approximation, the statistical characteristics of real systems. Each primitive element of a probabilistic automaton produces the "correct" output with probability $1-\epsilon$ and the "incorrect" output with probability ϵ. For example, a probabilistic "nan" element has the probabilistic truth-table of Fig. 7: it produces the "nan" function with probability $1-\epsilon$ and the "and" function with probability ϵ.

The matter of whether a primitive element is probabilistic or not is independent of whether it is slow or not. However, for the sake of simplicity we will consider only finite automata made from either of two sets of probabilistic primitives: the probabilistic version of the idealized primitives of Fig. 1, and the slow probabilistic primitives of Fig. 8. In each case the probabilistic operation of the primitives is defined with respect to a specified deterministic operation. The deterministic definition gives the desired or "correct" answer, and in discussing probabilistic automata it is often convenient to refer to this deterministic norm. Probabilistically the element gives the correct answer with probability $1-\epsilon$, the incorrect answer with probability ϵ. In view of what has just been said, the probabilistic elements of Fig. 8 may be defined by specifying their deterministic behavior. The second primitive is a "majority element" whose initial output is zero and whose output at time t+1 is one (active) if and only if two or more of

its inputs are active at time t. The third primitive of Fig. 8 produces a constant zero (inactive) output. For simplicity we will allow unlimited branching with each set of primitives.

A great deal may be learned about probabilistic automata by studying the input-free case. Consider the simple example of Fig. 9, where the initial state of the delay is left unspecified. Its deterministic state diagram is given in Fig. 10; whatever its starting state, it remains in that state for ever. Hence, as a deterministic system its memory span is infinite, though of course its memory capacity is finite. But probabilistically it behaves differently, as its probabilistic state diagram (Fig. 11) shows. Probabilistically it can make a transition from either state to either state. Consequently a finite probabilistic automaton has only a finite memory span, as we shall now show.

Consider the state diagram of an arbitrary input-free finite probabilistic automaton. When the automaton is in a given internal state σ_i it may pass from this to any other state σ_j with a non-zero probability p_{ij}. Let n be the number of internal states of the automaton. Then for each internal state σ_i we have $\sum_{j=1}^{n} p_{ij} = 1$, and there is an n by n probabilistic transition matrix M governing the transitions between states, with every element of M being non-zero. Let $\pi = [p_1, p_2, \cdots, p_n]$ be the probability vector of the initial state, where p_i is the probability that the automaton will start in state σ_i. It is clear from all this that the sequence of states of the automaton is a Markov chain, starting in some state at time zero in accordance with the probability vector π and making transitions according to the Matrix M.

A regular Markov chain is one in which some finite power of the transition matrix has only positive elements. Since all elements of M itself are positive, the sequence of internal states of a finite, probabilistic, input-free automaton is a regular Markov chain. For any regular transition matrix M there is a unique probability vector θ such that for any probability vector π,

$$\text{(1)} \qquad \lim_{t \to \infty} \pi \cdot M^t = \theta \quad .$$

Note now that as we vary the unit delays of the automaton between those whose initial outputs are zero and those whose initial outputs are one, while leaving the rest of the automaton unchanged, we are varying the initial vector π but leaving the transition matrix M unchanged. Hence this theorem tells us that the ultimate statistical character of the sequence of internal states of a probabilistic automaton is independent of the initial state. For example, whether the question mark of Fig. 9 is replaced by a zero or a one, in the limit the output is as likely to be a one as a zero. Hence the initial state of a probabilistic automaton has less and less influence on its behavior as time increases.

We can study the memory properties of a probabilistic automaton by investigating how fast an input-free probabilistic automaton forgets its initial state. Specifically, we ask how much information its present internal state gives us about its initial state. Without any essential loss of generality we may confine our attention to Fig. 9. Let O(t) mean that the output is one (active) at time t. Imagine that Fig. 9 is a black box, so that we do

not know which delay element of Fig. 1 is there. There are two hypotheses: h says that the question mark is to be filled in with a one, $\bar{h}$ that it is to be filled in with a zero. That is, $p[O(0)|h] = 1-\epsilon$ and $p[O(0)|\bar{h}] = \epsilon$. If Fig. 9 were deterministic, O(t) for any t would give us perfect information about the box: O(t) implies h, $\overline{O(t)}$ implies $\bar{h}$. Since Fig. 9 is probabilistic, O(t) gives us only statistical information about h and $\bar{h}$. To see exactly how much information it gives we will apply Bayes' theorem of inverse probabilities.

Let $P[h|O(t)]$ be the probability of hypothesis h conditionally on the observation O(t), and similarly for $P[h|O(t)]$. By Bayes' theorem

$$\frac{P[h|O(t)]}{P[\bar{h}|O(t)]} \quad \frac{P(h)}{P(\bar{h})} \quad \frac{P[O(t)|h]}{P[O(t)|\bar{h}]} \; . \tag{2}$$

i.e., the ratio of the posterior probabilities equals the ratio of the prior probabilities times the ratio of the degrees of prediction. The ratio of the posterior probabilities divided by the ratio of the prior probabilities is a measure of the amount of information the observation O(t) gives us about the hypothesis h. This amount of information can be computed as a function of π, M, and t. We will not make the calculation here, but will instead look at what happens in the limit. By (1)

$$\frac{\text{Limit } P[O(t)|h]}{t \to \infty \; P[O(t)|\bar{h}]} = 1 \tag{3}$$

Hence

$$\frac{\text{Limit } P[h|O(t)]}{t \to \infty \; P[\bar{h}|O(t)]} \cdot \frac{P(h)}{P(\bar{h})} = 1 \tag{4}$$

In other words, as t increases the observation O(t) gives us less and less information about the contents of the black box, and in the limit it gives us

no information. Hence a probabilistic automaton forgets more and more as time continues and in the limit remembers nothing.

The forgetting of an input-free finite probabilistic automaton can also be analyzed in terms of Shannon's information theory [11]. For this purpose we can think of the automaton (cf. Fig. 6) as being stretched out in space, so that it consists of noisy unit delays alternating with a perfect switch ad infinitum. The initial state of the automaton is then transmitted through this infinitely long channel. At time t it will have passed through t delay-switch combinations, i.e., through t noisy delays and through t deterministic recodings. The loss of information as a function of time can be calculated by means of information theory. The result will be the same as we obtained by means of Bayes' theorem: as t increases indefinitely the amount of information transmitted approaches zero.

The result just given is for memory in input-free probabilistic automata. But remembering is a special kind of computation (as we defined the term) and input-free automata are a special kind of automata, so one can see by analogy that a similar result holds for the computation of probabilistic automata generally: as time passes the probability of a mistake in computation rises, and in the infinite limit the computed answer is mere noise. Fortunately, from a practical point of view we are interested only in finite computations, that is, in computations which take a finite amount of time. With respect to finite computations the actual rate of degeneration of information is of interest, as well as techniques for retarding this loss of information and rendering errors less likely.

The late John von Neumann developed a method for synthesizing reliable automata from unreliable elements [12]. His method is for slow switches only, but by means of the algorithms of Section 4 we will extend it to cover arbitrary finite automata. Von Neumann's method involves replacing single lines driving single elements by multiplexed cables driving arrays of elements. As extended, it accomplishes roughly the following: given a probabilistic idealized automaton A_i it produces a multiplexed automaton A_m related to A_i as follows: deterministically, A_m is much more reliable than A_i.

We will build the multiplexed automaton from the primitives of Fig. 8, but these are first made into multiplexed organs: a restoring organ (Fig. 12) and a multiplexed-and-restored nan organ (Fig. 13). The inputs and outputs of these multiplexed organs are cables of N wires rather than single wires; N is called the multiplexing factor. For simplicity we have used a multiplexing factor of only three in Figs. 13 and 14, but in practice the multiplexing factor must be much larger; the multiplexing factor N must be sufficiently large to give the reliability desired and to warrant the use of statistical methods in analyzing the behavior and computation of the multiplexed automaton.

A finite automaton constructed from these two kinds of organs is called a finite multiplexed automaton. Its inputs and outputs are cables of size N. In the sequel we will compare the computation of idealized automata with the computation of multiplexed automata; in this comparison single wires of idealized automata are to be compared with cables of multiplexed automata.

A multiplexed automaton is so constructed that if it were deterministic all the wires of a given cable would be in the same state at the same moment of time. Since the automaton is probabilistic this will not be so, but if N is sufficiently large compared to ϵ and the design is suitable then it will be nearly so; more precisely, under these conditions the probability will be high that most of the wires of a given cable will be in the same state at any given moment. The activity level of a multiplexed cable is interpreted relative to a statistical fiduciary parameter Δ as follows. If $(1-\Delta)N$ or more wires are active (in state one), the cable is taken to represent a one; if ΔN wires or less are active, the cable is taken to symbolize a zero; and if the activity level falls between Δ and $(1-\Delta)$ the signal on the cable is not interpreted (i.e., is "nonsense").

A restoring organ consists of a randomizer driving a parallel array of N majority elements (Fig. 12). Each of the N wires of the input cable is split into three wires before entering the randomizer; the randomizer permutes these wires before they drive the majority elements. Deterministically a restoring organ merely delays the pulses of the input cable one unit of time. Probabilistically a restoring organ operates as follows. Let $(\beta)\{\gamma\}$ be the probability that an (input wire) {output wire} is in error at any given time t. Since the randomizer only permutes wires, β is also the error probability of a single input wire of a majority element. If the randomizer rearranges the wires in a sufficiently random fashion the error probabilities of the three wires going into a single majority element will be independent. A detailed analysis of cases shows that

(5) $$\gamma < \epsilon + 3\beta^2 \quad .$$

For suitable ϵ and β we will have $\gamma < \beta$, i.e., the probability of output error is less than the probability of an input error.

An important use of the restoring organ is as part of the multiplexed-and-restored nan organ (Fig. 13). This consists of a parallel array of nan-elements followed by a restoring organ. Deterministically, the multiplexed-and-restored nan organ performs the nan switching function on two input cables and then delays the output two units of time. Probabilistically it operates as follows. Let $(\alpha_1)\{\alpha_2\}$ be the probability that a wire of the (first) {second} input cable is in error at time t, and let $\alpha = \text{Max}\ (\alpha_1, \alpha_2)$. We consider how the error probability β of the input to the restoring organ depends on α. There are two main cases to consider.

First, the two input cables may come from the same source (i.e., the nan organ negates the input), in which case

(6) $$\beta < \epsilon + \alpha \quad .$$

Second, the two input cables may come from separate sources; these will be either constant sources (e.g., all zeros) or other organs. In the former case the errors on wires of different cables are assumed to be independent; in the latter case the input cables came from restoring organs, so that the error probabilities α_1, α_2 for the two input cables operate independently. Considering the different possible input states we get

(7) $\beta < \epsilon + \alpha^2$ (for the zero-zero input state)

(8) $\beta < \epsilon + \alpha$ (for zero-one or one-zero inputs)

(9) $\beta < \epsilon + 2\alpha$ (for one-one input)

Of the formulas (6) through (9) the last gives the greatest error. Combining (9) with (5) we have, for the multiplexed-and-restored nan organ, in the worst case,

(10) $$\gamma < \epsilon + 3(\epsilon+2\alpha)^2 \quad .$$

So, for suitable ϵ and α it is the case that $\gamma < \alpha$, i.e., the probability of an output error is less than the probability of an input error.

A comparison of formulas (5) and (10) shows that the latter sets a bound on the output error probability of a single wire as a function of the input error probability, for any organ in a finite multiplexed automaton. Hence it is relevant to examine the effect of iterating formula (10) on itself, starting with the initial case of $\alpha = \epsilon$. It turns out that for sufficiently small ϵ the output error probability γ is always bounded. For example, if $\epsilon < .01$ the output error probability of any multiplexed organ is bounded by 1.5ϵ, and if $\epsilon < .001$ the output error probability is bounded by 1.03ϵ.

A full statistical analysis requires a consideration of the distribution of errors on the cables. There is not space for this here, but von Neumann [12] has done it for the (in general) worst case of a restoring organ composed of nan elements.

Von Neumann considered only slow switches, and hence did not study nets with cycles. Since the randomizing networks are fixed throughout time and

are only pseudo-random, it is possible that higher-order errors will be amplified when signals go around cycles repeatedly. This problem needs investigation. To reduce such errors one can vary the structure of the randomizer from one part of the net to the other and one can also use more restoring organs than our design algorithms use.

To see intuitively what happens we focus attention on a single finite multiplexed automaton and consider the effect of varying the multiplexing factor N or this automaton. Let Θ be the probability that the signal level in a given cable will fall inside the "nonsense" band Δ to $(1-\Delta)$. By increasing N indefinitely we can bring Θ arbitrarily close to zero. Since every finite multiplexed automaton contains a finite number of organs it follows that by increasing N indefinitely we can increase indefinitely the mean free path between errors in the automaton and hence increase indefinitely the reliability of the automaton.

This increase in reliability is purchased, of course, at the cost of more equipment. For the cases he considered, von Neumann [12] found large multiplexing factors (e.g., 20,000) to be necessary, but two points should be noted in this connection. First, in order to treat an extreme case we have deliberately chosen a restoring primitive which requires large multiplexing factors for reasonable reliability. (Compare in this respect our choice of the slow primitives of Section 4.) There are restoring primitives which give much better results.

For example, a generalized majority element with 2n+1 inputs, whose output is active if and only if n+1 or more inputs are active. As n increases

this becomes a better restoring primitive [13].

Second, the algorithms to be given next apply to arbitrary finite automata and for this reason will not give as good results as a procedure designed for a special class of automata (e.g., a computer organized into memories, arithmetic units, transmission lines, etc.).

We will now adapt our earlier algorithms for slow automata (Section 4) to finite multiplexed automata. To begin with, look at the multiplexed organs deterministically and in terms of operations on cables (rather than on single wires). Looked at this way the restoring organ is a unit delay and the multiplexed-and-restored nan organ is a nan-switch followed by two units of delay. There is also available the constant zero (i.e., a cable all of whose wires are inactive). From these logical elements we can synthesize all the slow primitives of Fig. 5, with the cables of the multiplexed automaton doing the job performed by the wires of the slow automaton.

In this manner we obtain two more algorithms. When given a finite idealized automaton A_i and a desired mean free path between errors r, our third algorithm designs a finite multiplexed automaton A_m with these properties: deterministically A_m is computationally equivalent to A_i, probabilistically the mean free path between errors in A_m is r. When given a finite idealized automaton A_i whose read control output is ultimately periodic and a desired mean free path between errors r, our fourth algorithm designs a multiplexed automaton A_m^* with these properties: deterministically A_m^* produces the same computation as A_i and at the same rate (after an initial delay), probabilistically the mean free path between errors in A_m^* is r.

It is of interest to compare A_i, A_m, and A_m^*, since deterministically they are all computationally equal. A_m has many more primitive elements than A_i and computes more slowly than A_i, but probabilistically it is more reliable than A_i. Here reliability is bought at the cost of speed and equipment. A_m^* is even larger than A_m, but computes at the same rate as A_i. Here reliability is bought at the cost of equipment (and a fixed initial delay). As in the case of slow automata, it is desirable to have some general theorems governing interchanges of speed, equipment, and reliability of the kind just illustrated.

6. Cellular Automata

Slow automata (Section 4) take account of the time required for switching and the limit on the number of wires that a single component can drive. In actual computers there are also limits on the length of wires and there are design problems which arise in connection with the geometrical arrangement of components. These factors are partially taken account of on a theoretical level by means of cellular automata. A cellular automaton is a finite or infinite array of cells, each cell containing a finite automaton which is connected to certain "neighbors." The infinite shift register (tape) of a Turing machine (Sections 2, 4) is a one-dimensional infinite cellular automaton.

In an unfinished manuscript von Neumann [14] developed an infinite two-dimensional cellular model with square cells. Each cell is occupied by the

same 29 state finite automaton

There is a fiducial initial state U. Initially all but a finite number of cells must be in this state.

---

which communicates directly with its four immediate neighbors with a delay of one unit or more. Though von Neumann did not finish the construction, he carried it far enough to show that both a Universal Turing machine and a self-reproducing automaton can be embedded in this model.

One can also study finite cellular automata by considering some finite set of cells and taking the inputs and outputs from along the edges. The problems of Section 4 ("Slow Automata") and, if probability is introduced, of Section 5 ("Probabilistic Automata") then arise. There is time to mention only one result here. There is an algorithm, analogous to our first algorithm, which does the following: given a finite idealized automaton A_i, this algorithm produces a finite von Neumann cellular automaton A_c which is computationally equivalent to A_i [15].

References

[1] Turing, A. M. "On Computable Numbers, with an Application to the Entscheidungsproblem." Proceedings of the London Mathematical Society Series 2, 42 (1936) 230-265 and 43 (1937) 544-546.

[2] Moore, E. F. "Gedanken-Experiments on Sequential Machines." Pp. 129-153 of Automata Studies (edited by C. E. Shannon and J. McCarthy), Princeton University Press, 1956.

[3] Burks, A. W. and Hao Wang. "The Logic of Automata." Journal of the Association for Computing Machinery 4 (April, 1957) 193-218) and 4 (July, 1957) 279-297.

[4] Burks, A. W. "Computation, Behavior, and Structure in Fixed and Growing Automata." Pp. 282-311 of Self-Organizing Systems (edited by Marshall Yovits and Scott Cameron), Pergamon Press, New York, 1960.

[5] Yamada, H. "A Mode of Real Time Operations of a Subclass of Turing Machines and the Existence of a Subclass of Recursive Functions Which are not Real Time Computable," to appear in Transactions of IRE, EC.

[6] McNaughton, Robert. "The Theory of Automata, a Survey." to appear in Advances in Computers, Vol. II (edited by Franz L. Alt), Academic Press.

[7] Friedman, Joyce. "Some Results in Church's Restricted Recursive Arithmetic." Journal of Symbolic Logic 22 (Dec., 1957) 337-342.

[8] Friedman, Joyce. "A Decision Procedure for Computations of Finite Automata." Unpublished manuscript.

[9] Burks, A. W. and Jesse B. Wright. Sec. 4.1 of "Sequence Generators and Digital Computers." to be published in the Proceedings of the Symposium on Recursive Function Theory (April, 1961, N.Y.C.).

[10] McNaughton, Robert. "On Nets Made up of Badly Timed Elements. Part I: Slow But Perfectly Timed Elements." Mimeographed, 30 pages. The Moore School of Electrical Engineering, University of Pennsylvania, 1961.

[11] Shannon, Claude E. "A Mathematical Theory of Communication." The Bell System Technical Journal 27 (1948) 379-423 and 623-656.

[12] Von Neumann, John. "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components." Pp. 43-98 of Automata Studies, op. cit.

[13] Verbeek, L.A.M. "On Error Minimizing Neural Nets." Pp 121-123 of Principles of Self-Organization (edited by H. von Foerster and G. W. Zopf) Pergamon Press, New York, 1962.

[14] Von Neumann, John. "The Theory of Automata: Construction, Reproduction, Homogeneity." To be published by the University of Illinois Press under the editorship of the present writer.

[15] Burks, A. W. "Cellular Automata." To be published in the Proceedings of the International Symposium on Theory of Relay Systems and Finite Automata held in Moscow, Sept. 24-Oct. 2, 1962.

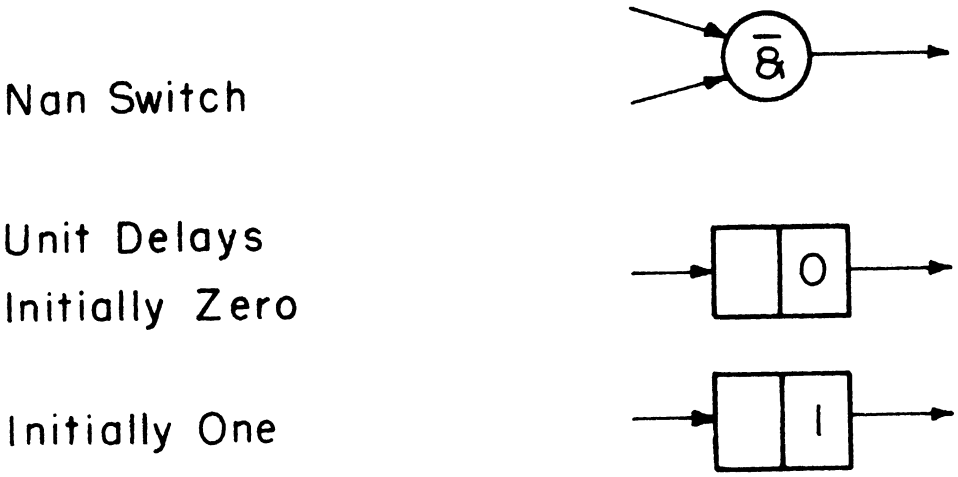

Fig. 1. Idealized primitives.

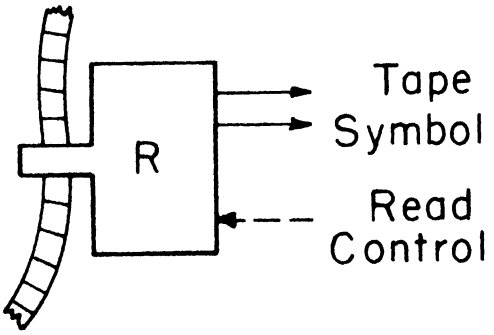

Fig. 2. Input tape reader.

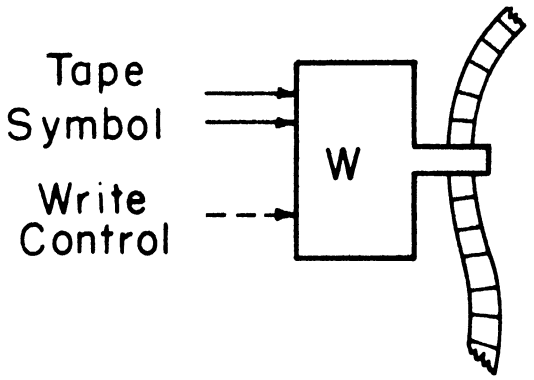

Fig. 3. Output tape writer.

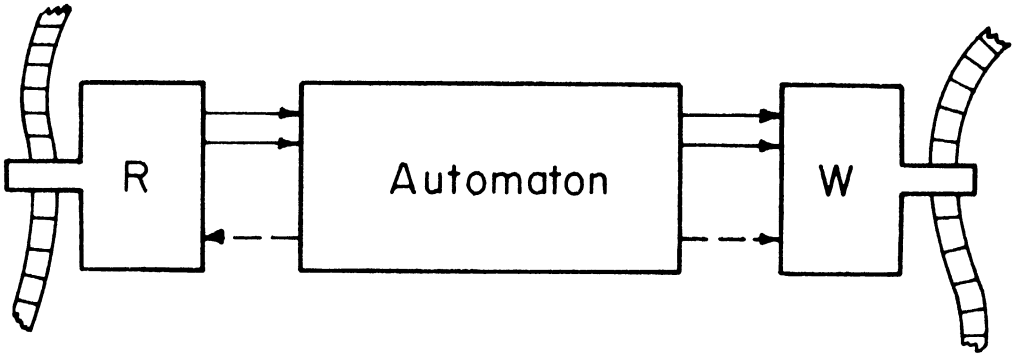

Fig. 4. Computation of an automaton.

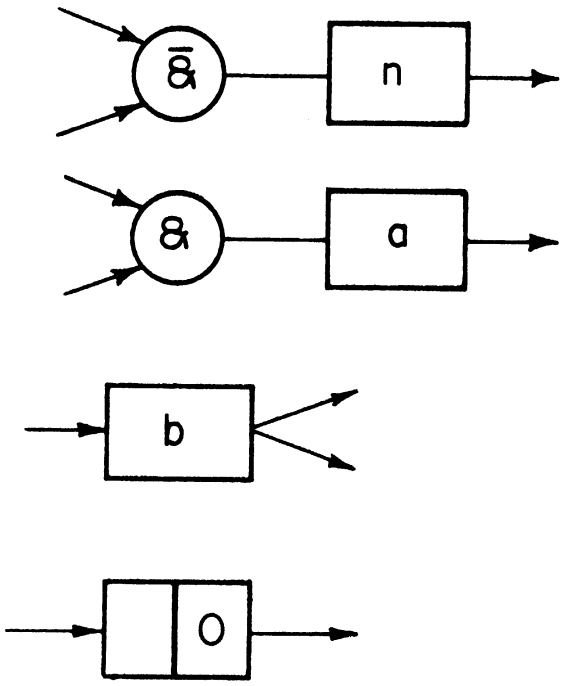

Fig. 5. Slow primitives.

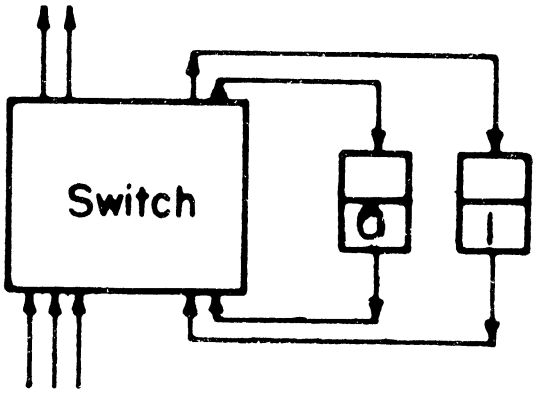

Fig. 6. Normal form net.

Input State		Output with Probability	
		$1-\epsilon$	ϵ
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 7. Probabilistic nan.

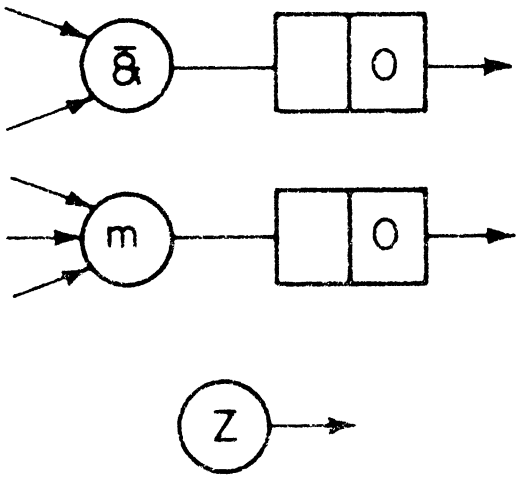

Fig. 8. Slow probabilistic primitives.

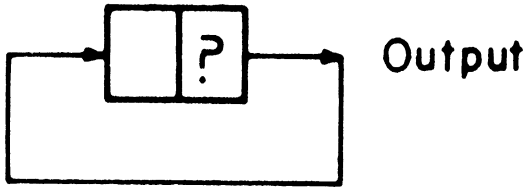

Fig. 9. Probabilistic automaton.

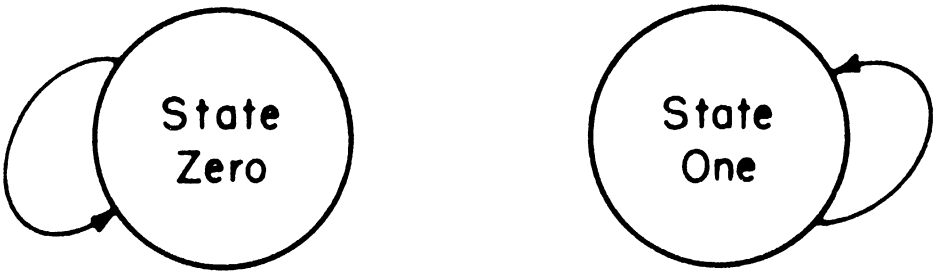

Fig. 10. Deterministic state diagram.

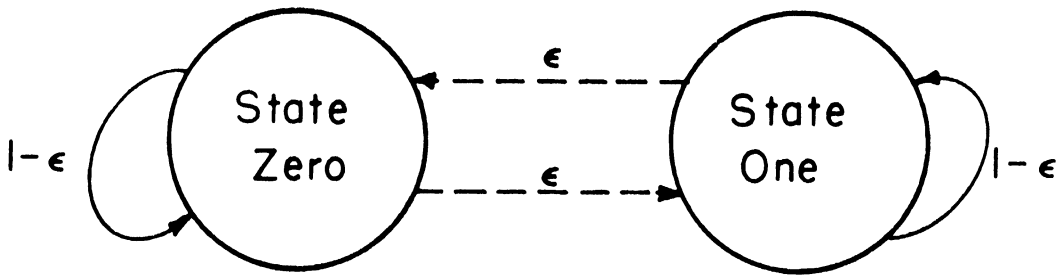

Fig. 11. Probabilistic state diagram.

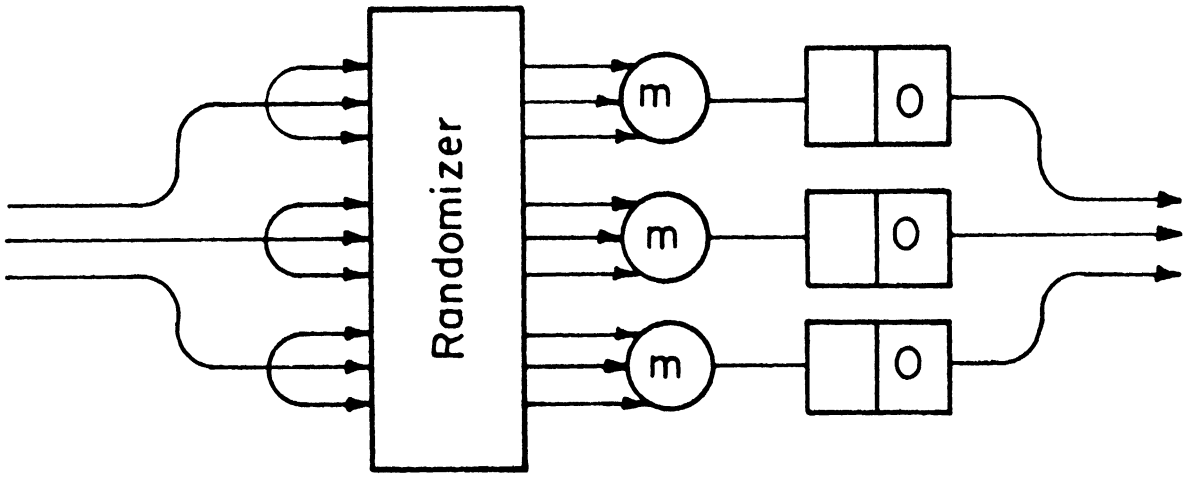

Fig. 12. Restoring organ—multiplexing factor N = 3.

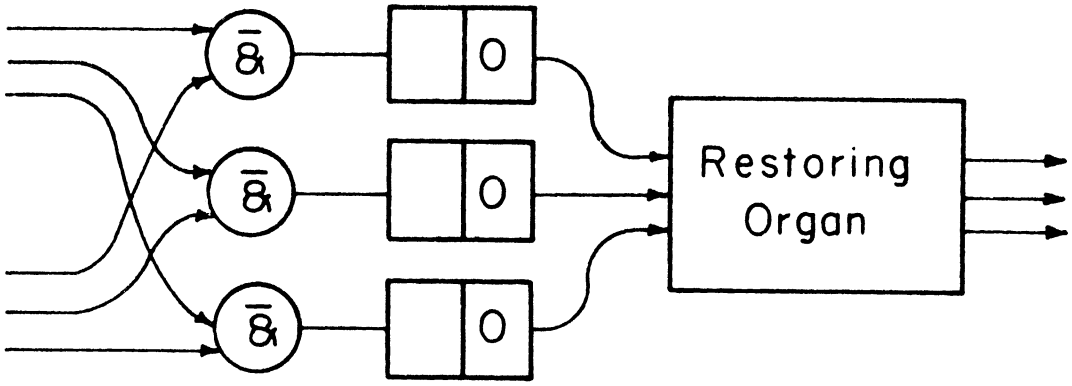

Fig. 13. Multiplexed-and-restored nan organ—multiplexing factor N = 3.

NERVE-NET SIMULATION

J. Willison Crichton
Marion R. Finley

This quarter was devoted to completing the adaptation of the nerve net program to The University of Michigan Computing Center's IBM 709 (and later, IBM 7090) system. The program had been written originally in the SCAT language for The University of Michigan Willow Run Laboratories IBM 709 system and debugged to the point where some vital changes in the underlying statistical analysis were found to be necessary and some experiments on sustained activity were carried out with positive results. From the time of the transfer of the IBM 709 from the Willow Run Laboratories to the Computing Center until August 1962, the main efforts were directed towards adapting the program to the University of Michigan system. The authors were forced to write their own assembly program, called SCALP, which is specially adapted to the needs of the simulation program, and yet operates within the procedures of The University of Michigan Computing Center's system. After converting the simulation program to SCALP, success was immediately obtained in debugging the input-output and tape handling routines and simultaneously runs on the three-neuron correlation experiment and a "shake-down" test of the central net program were begun. There was a gap in computer operation from August 28 until September 24 during which period the Computing Center was closed to replace the IBM 709 with the IBM 7090. This period was used by the authors to do additional preparation for future runs. After installation of the IBM 7090 was completed (which caused only minor changes in the simulation program), the runs above were continued. In early October, a complete interpretive trace routine with a monitoring option was written and debugged to allow detailed examination of

the net program as an aid in tracking down the source of any anomalous behavior.

The writing and debugging of the SCALP routine mentioned above took place in July and August. A write-up on SCALP is available (SCALP—Symbolic Compressed Assembly and Loading Program by the same authors); a brief description follows:

The SCALP system performs a minimal assembly and loading function. It does not produce a new deck of cards of any kind. It does not possess any powerful translation features such as macro translation. It does not produce the kind of detailed listing produced by the SOS assembler of The University of Michigan assembly program (UMAP). It does not make automatic provision for multiple core-load programs. However, for those who are obliged to write rather complex programs in machine language, it does possess the following advantageous features:

(1) It is simple to learn and has few pitfalls.

(2) It performs a very fast assembly relative to more elaborate systems.

(3) It allows the writing of several instructions per card, thus permitting a long program to be contained in a rather small symbolic deck.

(4) It does not require a strict card format, in the sense that certain things must be punched in certain columns. The format requirements are contextual rather than absolute.

(5) It provides optional linkage with The University of Michigan Executive System and I-O Supervisor.

(6) It provides for the inclusion of relocatable binary subroutines.

The aim of SCALP is to make it convenient, and also feasible from the standpoint of machine time, to use the original symbolic deck as the permanent form in which a program is handled.

PERSONNEL

The following is a summary of the hours worked under this contract for the period 15 July to 15 October 1962.

A. W. Burks	205
J. H. Holland	368
J. W. Crichton	336
M. R. Finley	496
J. W. Thatcher	96
R. A. Laing	96
Total man hours	1597

OBJECTIVES FOR NEXT QUARTERLY PERIOD

The research in environmental regularities (opportunities for the adaptive system to depart from random or enumerative behavior) which was begun in this last quarter, will be continued. This research topic will be developed in detail for the more particular "tree-search" case (an environment, specified by a tree with assigned utilities, which also includes games, search procedures, etc.). Specific theorems relating the limiting adaptive efficiency to the regularities present will be the longer range goal, with the immediate objective being to establish the maximum possible rate of adaptation when the utilities are assigned to the tree at random (the "absence of regularities" case). (Holland).

In the nerve-net simulation experiments, the next quarterly period will be devoted to completion of the debugging of the main program, and initiating the planned series of experiments. A simple macro-instruction feature will be incorporated into the SCALP system, with a view to reducing the rate of errors in the programming of experiments (Finley and Crichton).

DISTRIBUTION LIST

(One copy unless otherwise noted)

OASD (R&E), Room 3E1065
The Pentagon
Washington 25, D.C.
Attn: Technical Library

Chief, Research and Development
OCS, Department of the Army
Washington 25, D.C.

Commanding General
U. S. Army Materiel Command
Washington 25, D.C.
Attn: R&D Directorate

Commanding General 3
U. S. Army Electronics Command
Fort Monmouth, N.J.
Attn: AMSEL-AD

Commander 10
ASTIA
Arlington Hall Station
Arlington 12, Virginia
Attn: TIPCR

Commanding General
USA Combat Developments Command
Fort Belvoir, Virginia
Attn: CDCMR-E

Commanding Officer
USA Communication and Electronics
Combat Development Agency
Fort Huachuca, Arizona

Chief, U.S. Army Security Agency 2
Arlington Hall Station
Arlington 12, Virginia

Deputy President
U.S. Army Security Agency Board
Arlington Hall Station
Arlington 12, Virginia

Director
U.S. Naval Research Laboratory
Washington 25, D.C.
Attn: Code 2027

Commanding Officer and Director
U.S. Navy Electronics Laboratory
San Diego 52, California

Aeronautical Systems Division
Wright-Patterson Air Force Base, Ohio
Attn: ASAPRL

Air Force Cambridge Research
Laboratories
L. G. Hanscom Field
Bedford, Massachusetts
Attn: CRZC
Attn: CRXL-R

Hq., Electronic Systems Division
L. G. Hanscom Field
Bedford, Massachusetts
Attn: ESAT

APSC Scientific/Technical Liaison
Office
U.S. Naval Air Development Center
Johnsville, Pennsylvania

Commanding Officer
U.S. Army Electronics Materiel
Support Agency
Fort Monmouth, New Jersey
Attn: SELMS-ADJ

Director, Fort Monmouth Office
USA Communication and Electronics
Combat Development Agency
Fort Monmouth, New Jersey

DISTRIBUTION LIST (Concluded)

Corps of Engineers Liaison Office
U.S. Army Electronics Research
and Development Laboratory
Fort Monmouth, New Jersey

Marine Corps Liaison Office
U.S. Army Electronics Research
and Development Laboratory
Fort Monmouth, New Jersey

Commanding Officer
U.S. Army Electronics Research
and Development Laboratory
Fort Monmouth, New Jersey
Attn: Logistics Laboratory
Attn: Director, Research/Engineering
Attn: Technical Documents Center
Attn: SELRA/SL-NPE 4
Attn: Technical Information Div. 3
Attn: Mr. Anthony V. Campi 14

Commanding General
U.S. Army Materiel Command
Washington 25, D.C.
Attn: R&D Directorate
Research Division
Electronics Branch

Professor Heinz Von Foerster
215 Electrical Engineering Research
Laboratory
University of Illinois
Urbana, Illinois

Dr. Saul Amarel
RCA Laboratories
Princeton, New Jersey

IBM Research Laboratory
Bethesda, Maryland
Attn: Dr. A. Babbitt

Sterling Perkes
Department of Electrical Engineering
University of Utah
Salt Lake City, Utah