THE UNIVERSITY OF MICHIGAN

COLLEGE OF LITERATURE, SCIENCE, AND THE ARTS
Department of Philosophy

Technical Report

ITERATIVE CIRCUIT COMPUTERS

John Holland

UMRI Project 2794

under contract with:

DEPARTMENT OF THE ARMY
U. S. ARMY SIGNAL CORPS
CONTRACT NO. DA-36-039-SC-78057
FORT MONMOUTH, NEW JERSEY

administered by:

THE UNIVERSITY OF MICHIGAN RESEARCH INSTITUTE ANN ARBOR

May 1960

This research was supported by the U.S. Army Signal Corps through contract DA-36-039-SC-78057, and by the National Science Foundation through grants G-4790 and G-11046. The paper was written while the author was a member of the Institute of Science and Technology at The University of Michigan.

SUMMARY

The paper first discusses an example of a computer, intended as a prototype of a practical computer, having an iterative structure and capable of processing arbitrarily many words of stored data at the same time, each by a different sub-program if desired. Next a mathematical characterization is given of a broad class of computers satisfying the conditions just stated. Finally the characterization is related to a program aimed at establishing a theory of adaptive systems via the concept of automaton generators.

I

INTRODUCTION

Computers constructed of hundreds of millions of logic and storage elements will require an organization radically different from present computers if the elements are to be used efficiently in computation. It should be possible to process arbitrarily many words of stored data at the same time, each by a different sub-program if desired. In addition the structure of the computer should be iterative or modular in order to allow efficient use of template techniques in its construction.

The present paper contains a mathematical characterization of a broad class of computers satisfying these conditions. This class, with appropriate interpretation of the symbols, includes representatives structurally and behaviorally equivalent to each of the following types of automata:

(1) Turing machines (with 1 or more tapes) [12,9]

(2) Tessellation automata (Von Neumann, Moore)[14,7]

(3) Growing logical nets (Burks-Wang)[2,1]

(4) Potentially-infinite automata (Church)[3]

The class also contains automata which, in various senses, are generalizations of each of these four types.

Ultimately, for the designer, the value of such a characterization depends upon whether or not it can actually suggest designs for solid-state computer. Section II of this paper discusses a possible abstract prototype for such a computer—it is one of many alternatives which can be defined and investigated with the help of the characterization. (A similar computer is discussed in greater detail in the Proceedings of the 1959 Eastern Joint Computer Conference)[5]. Section III summarizes the mathematical characterization and considers its interpretation. Section IV relates this paper to a program (begun by the author in 1958) which has as its objective a theory of adaptive systems.

II

AN ITERATIVE CIRCUIT COMPUTER

The computer outlined in this section is presented primarily to suggest something of the class of computers included in the characterization summarized in the next section. At the time, however, the order code, addressing schemes, etc. were chosen to reflect their counterparts in present computers. In this sense, the computer can also be thought of as a prototype of a practical computer—assuming that the large numbers of components required can be provided economically. Because the computer can execute an arbitrary number of sub-programs simultaneously, and because the sub-programs are spatially organized, its operation is of course considerably different from present computers.

The computer can be considered to be composed of modules arranged in a 2-dimensional rectangular grid; the computer is homogeneous (or iterative) in the sense that each of the modules can be represented by the same fixed logical network. The modules are synchronously timed and time for the computer can be considered as occurring in discrete steps, $t = 0, 1, 2, \ldots$

Basically each module consists of a binary storage register together with associated circuitry and some auxiliary registers. At each time-step a module may be either active or inactive. An active module, in effect, interprets the number in its storage register as an instruction and proceeds to excute it. There is no restriction (other than the size of the computer) on the number of active modules at any given time. Ordinarily if a module $M(i,j)$ at coordinates (i,j) is active at time-step t, then at time-step $t+1$, $M(i,j)$ returns to inactive status and its successor, one of the four neighbors $M(i+1,j)$, $M(i,j+1)$, $M(i-1,j)$, or $M(i,j-1)$, becomes active. (The exceptions to this rule occur when the instruction in the storage register of the active module specifies a different course of action as, for example, when the instruction is the equivalent of a transfer instruction).

The successor is specified by bits s_1, s_2 in M(i,j)'s storage register. If we define the line of successors of a given module as the module itself, its successor, the successor of the successor, etc., then a given sub-program in the computer will usually consist of the line of successors of some module. Since several modules can be active at the same time the computer can in fact execute several sub-programs at once. We have noted parenthetically that there are orders which control the course of action—there are also orders equivalent to store orders which can alter the number (and hence the instruction) in a storage register. Therefore, the number of sub-programs being executed can be varied with time, and the variation can be controlled by one or more sub-programs.

The action of a module during each time-step can be divided into three successive phases:

(1) During phase one, the initial phase of each time-step, a module's storage register can be set to any arbitrarily chosen value and its auxiliary registers to any desired condition. The numbers and conditions thus supplied are the computer's input. Although the number in the storage register can be arbitrarily changed at the beginning of each time-step, it need not be; for many purposes the majority of modules will receive input only during the first few moments of time ("storing the program") or only at selected times t_1, t_2, ... ("data input"). Of course, some modules may have a new number for input at each time-step; in this case the modules play a role similar to the inputs to a sequential circuit.

(2) During phase two, an active module determines the location of its operand set, the set of storage registers upon which its instruction is to operate. This the module does by, in effect, opening a branching path (sequence of gates) to the operands. The path-building action depends upon two properties of modules:

First by setting bit p in its storage register equal to 1, a module may be given special status which marks it as a point of origination for paths; the module is then called a P-module.

Secondly, each module has a neighbor, distinct from its successor, designated as its predecessor by bits q_1, q_2 in its storage register; the line of predecessors of a given module M_0 is then defined as the sequence of all module [M_0, M_1,.., M_k, ...] such that, for each k, M_k is the predecessor of M_{k-1} and M_{k-1} is the successor of M_k. Note that the line of predecessors may in extreme cases be infinitely long or non-existent. The line of predecessors of an active module ordinarily serves to link it with a P-module (through a series of open gates). During the initial part of phase two the path specification bits $y_0,\dots,y_n$ and $d_0,\dots,d_3$ in the storage register of an active module M_0, are gated down its line of predecessors to the nearest P-module (if any) along that line. The path specification bits are then used by the P-module to open a branching path to the operand set of the active module.

Each path must originate at a P-module. The modules belonging to a given path can be separated into sub-sequences call segments. Each segment consists of y modules extending parallel to one of the axes from some position (i,j through positions $(i+b_1, j+b_2)$, $(i+2b_1, j+2b_2)$, ..., $(i+(y-1)b_1, j+(y-1)b_2)$, where $b_1 = \pm 1$ or 0 and $b_2 = \pm (1-b_1)$; the module at $(i+yb_1, j+yb_2)$ will be called the termination of the segment. Each module possesses four *-registers and if the module belongs to a segment in direction (b_1,b_2) the appropriate *-register, $(b_1,b_2)*$, is turned on gating lines between (i,j) and $(i+b_1, j+b_2)$. Since each *-register gates a separate set of lines, a module may (with certain exceptions) belong to as many as four paths. Once a *-register is turned on it stays on until it is turned off; thus a path segment, once marked, persists until "erased".

Each segment of a path results from the complete phase two action of a single active module; however, since a path may branch, more than one segment may result in one time-step from the action of a given active module. After the digits $y_n,\ldots,y_o$, $d_3,\ldots,d_o$ are gated to the nearest P-module along the line of predecessors of the active module, new segments are constructed at the termination of each branch of the path originating at the P-module. Note that, because of the branching, there will be more than one path termination.

Branching is controlled by the digits $d_3,\ldots,d_o$. To each of the four digits $d_3,\ldots,d_o$ corresponds one of the four neighbors at each branch termination. If $d_i = 1$ then, when the path is extended, at each existing path termination a new branch will be sent through the i^{th} neighbor parallel to the axis.

Path extension takes place only when bit $y_n = 0$; then bits $y_{n-1},\ldots,Y_o$ determine the common length of thenew segments and bits $d_3,\ldots,d_o$ determine their directions. If $y_n = 1$ then final path segments, if any, in the directions specified by $d_3,\ldots d_o$ are erased (bits $y_{n-1},\ldots,y_o$ not being used in this case). In order to prevent interference of one path with another, or with itself, a set of priority and interlock rules are required. These rules will not be specified here but the interested reader can see a complete set of such rules for a similar computer in the 1959 E.J.C.C. paper cited previously.[5]

(3) During phase three, an active module executes the instruction contained in its storage register. This involves the following modules: the active module itself holds the order code in bits i_2, i_1, i_o of its storage register; the storage registers of the modules terminating the nearest path contain the set of words to be operated on (the operand set); finally there must be a module which serves as arithmetic unit. In order to serve as an arithmetic unit, bits (p,a) in the storage register of a module must first be set to the value

(0,1), giving the module special staus—A-module status. (Note that this means a module in P-module status, $p = 1$, cannot be an A-module). If $M(i,j)$ is an active module then the first A-module along its line of predecessors serves as the arithmetic unit.

A short, though representative, set of orders follows:

(i) Execution of OR/ADD causes the following sequence of actions; first the numbers stored at the modules in the operand set are transferred down the branches of the path toward the P-module; as these numbers meet at branch-points a resultant is formed equal to the bit-by-bit disjunction of the incoming numbers (i.e. bit j in the resultant is 1 only if at least one of the incoming numbers has 1 at position j); when the final resultant is formed at the P-module it is transferred along the line of predecessors to the nearest A-module; there the number is added to whatever number is in the storage register of the A-module. Note that this sequence of actions takes place wholly within phase three of the time-step in which the instruction is executed.

(ii) Execution of AND/ADD proceeds just as OR/AND except that a bit-by-bit conjunction (output bit is 1 only if all corresponding input bits are 1) takes the place of bit-by-bit disjunction.

(iii) Execution of STORE causes the number in the storage register of the nearest A-module to be transferred to the storage registers of all modules in the operand set.

(iv) Execution of TRANSFER ON MINUS depends upon the number in the storage register of the nearest A-module. If, in this number, $y_n = 0$ then at the end of phase three the active module becomes inactive and its successor become active. If $y_n = 1$ then each of the modules in the operand set, rather than the successor, become active.

(v) NO ORDER causes the execution phase to pass without the execution of an order.

(vi) STOP causes the active module to become inactive without passing activity on to its successor at the next time-step.

With the exception of the TRANSFER and STOP orders, the active module becomes inactive and its successor becomes active at the conclusion of phase three. Just as in the case of phase two some rules are required to prevent interference of active modules and to provide for cases where there is no nearest A- or P-module along the line of predecessors (the reader is again referred to the 1959 E.J.C.C. paper)[5].

The storage register of each module in the present formulation consists of n + 14 bits labelled in the following order:

bit number:
n + 14 n+ 13...14 13 12 11 10 9 8 7 6 5 4 3 2 1

label:
y_n $y_{n-1}\ldots y_0$ d_3 d_2 d_1 d_0 i_2 i_1 i_0 s_1 s_2 q_1 q_2 p a

The function, in the active module, of each bit group has already been discussed.

III

MATHEMATICAL CHARACTERIZATION OF ITERATIVE CIRCUIT COMPUTERS

One purpose of the mathematical characterization summarized here is to define the class of iterative circuit computers precisely enough to allow mathematical deduction to be used in their study. This property of the characterization will be used in later work in an attempt at establishing a theory of adaptive systems (see the next section). At the same time the characterization can be used to generate a wide range of computer prototypes, each with different structural and operational characteristics. Thus, the characterization can also be of help in the design of solid-state computers.

The characterization is made up of the following parts:

(1) The positions of the modules are indexed by the elements of a finitely-generated abelian group, A. The particular group chosen determines the "geometry" of the network; for instance, by choosing the appropriate group, the modules can be arranged in a plane, or a torus, or an n-dimensional cube, etc. Thus, for a computer with the modules arranged in a 2-dimensional rectangular grid 1000 modules on a side, A would be the abelian group with two generators a_1, a_2 satisfying the relations

$$\begin{aligned} 1000\, a_1 &= e \\ 1000\, a_2 &= e \end{aligned}$$

where e is the identity element of the group.

The group, A, is restricted to being a finitely-generated abelian group for several technical reasons. One reason is that the elementary theory of such groups is decidable. When taken with the rest of the definition of iterative circuit computers, this implies that the operation of the computer is effectively defined. Also any such group can be decomposed into a direct product of cyclic subgroups. Thus the elements of the group can be represented uniquely as n-tuples on the basis of certain sets of generators of the group (in other words, the modules are arranged in a "regular" fashion).

(2) In order to determine the immediate neighbors of a module we must specify a finite set, $A° = (a_1,\ldots,a_k)$, of elements selected from the group A. Then the set of immediate neighbors of the module at $\alpha\epsilon\{A\}$ are the modules at $a_i(\alpha) = \alpha + a_i$ for all $a_i \epsilon A°$, where + is the group operation. For example, if we have a module at coordinates (i,j) relative to generators a_1, a_2 and we wish its immediate neighbors to be at coordinates (i+1, j), (i, j+1), (i-1, j), and (i,j-1) then we could choose $A° = \{a_1,a_2,a_1^{-1},a_2^{-1}\}$, where a_i^{-1} is the group inverse of a_i.

(3) The state of <u>each</u> module in the computer at each time t must be drawn from a finite set, S, of allowable states. $S = X \otimes Y$, the cartesian product of the sets X and Y. X can be any finite set of elements—the elements will be called "storage states"; $Y = \prod_{i=1}^{k} Y_i$, a cartesian product of the sets $Y_i = R = \{a_1, \phi\} \otimes \ldots \otimes \{a_k, \phi\}$.

In what follows the notation S_α^t will be used to denote the state of the module at position α at time t. A similar convention will be used for the components of S. Note that the elements of Y can be though of as k by k matrices. The matrix Y_α^t which holds for the module α at time t is called the connection matrix of α at time t. The ith row Y_α^t, $Y_{\alpha i}^t = (Y_{\alpha i1}^t, Y_{\alpha i2}^t,\ldots,Y_{\alpha ik}^t)$, specifies which of the k immediate neighbors of α are connected through α to the module at $a_i^{-1}(\alpha)$; if $y_{\alpha ij}^t = a_j$ then the module at $a_j(\alpha)$ is connected through α to the module $a_i^{-1}(\alpha)$ otherwise not.

(4) Changes in the k rows $Y_1,\ldots,Y_k$ of the connection matrix Y from one time-step to the next, $Y_{\alpha i}^t$ to $Y_{\alpha i}^{t+1}$, are determined by a set of k projections:

$$P_i : S \to R = \{a_1,\phi\} \otimes \ldots \otimes \{a_k, \phi\}.$$

The way in which these changes are effected will be described in terms of the transition equations to be given shortly.

(5) Change in the storage state from one time-step to the next, X_α^t to X_α^{t+1}, is determined by a function

$$f : \prod_{i=1}^{k} S \to X$$

which will be called the "sub-transition function". Again f can be best explained in terms of the transition equations.

A particular selection for each of the five parts described in 1) through 5), $(A, A°, X, \{P_i\}, f)$, determines a particular iterative circuit computer. In addition a function $B : \{A\}\ \{t\} \to S$ may be effectively defined for some pairs (α,t); B_α^t gives the input to α at time t, when B is defined for (α,t). Once a particular iterative circuit computer is specified (and its initial state is given), the transition equations together with the function B determine the operation of the computer.

The transition equation for the connection matrix Y^t_α in terms of its elements, $Y^t_{\alpha ij}$, is:

$$Y^{t+1}_{\alpha ij} = \emptyset \qquad , \text{ if } P^t_{\alpha ij} = \emptyset \qquad (\text{"erasure"})$$

$$= Y^t_{\alpha ij} \qquad , \text{ if } P^t_{\alpha ij} = a_j \text{ and } Q^t_{\alpha ij} = 0 \quad (\text{"no change"})$$

$$= P^t_{\alpha ij} \qquad , \text{ if } P^t_{\alpha ij} = a_j \text{ and } Q^t_{\alpha ij} = 1 \quad (\text{"construction"})$$

where $P^t_{\alpha ij}$ is the jth component of $P_i(S^t_\alpha)$

$$\text{and } Q^t_{\alpha ij} = \&(q^t_{a_j(\alpha)j1},\ldots,q^t_{a_j(\alpha)jk})$$

$$q^t_{\beta ij} = 0, \text{ if } Y^t_{\beta ij} = \emptyset \text{ and } P^t_{\beta ij} = a_j$$

$$= 1, \text{ if } P^t_{\beta ij} = \emptyset$$

$$= \&(q^t_{a_j(\beta)j1},\ldots,q_{a_j(\beta)jk}), \quad \text{otherwise}$$

$$\text{defining } \&(c_1,\ldots,c_k) = 1, \text{ if all } c_j = 1$$

$$= 0, \text{ otherwise.}$$

As mentioned in (3) above, row i of the connection matrix Y^t_α can be interpreted as specifying a set of modules a (α) connected through α to $a_i^{-1}(\alpha)$; for each such j, row j of the connection matrix $Y_{a_j}(\alpha)$ may specify other modules $a_h a_j(\alpha)$ connected, via $a_j(\alpha)$ and then α, to $a_i^{-1}(\alpha)$; appropriate rows of the matrices $Y_{a_h a_j}(\alpha)$ may specify still others; etc. In other words the matrix Y^t_β tells how information is to be channeled through β to its immediate neighbors, the matrices for these neighbors tell how the information is to be sent on from there, etc. In this way each module serves as the base of what may be a complex branching tree channeling information to it. It will be seen (in the transition equations for X^t_α) that modules belonging to the tree for α pass information to α <u>without</u> a time-step delay.

The transition equation for Y^t_α can now be given the following interpretation: Broadly, $P^t_{\alpha i}$ (i.e. $P_i(S^t_\alpha)$) specifies changes in row i of the connection matrix Y_α^t while $Q_{\alpha i}^t = (Q_{\alpha j 1}^t,\ldots,Q_{\alpha ik}^t)$ prohibits certain of these changes. More specifically, $Q_{\alpha ij}^t = 0$ prohibits construction of a new connection from $a_j(\alpha)$ through α to $a_i{}^{-1}(\alpha)$ at time t. $Q_{\alpha ij}^t = 0$ just in case construction of a new connection is indicated somewhere in the tree for α. Some thought will show that this rule (others, at least superficially more general, could have been chosen) implies the following desirable conditions:

(i) a cycle of connections without delay cannot be formed (operation of modules belonging to such a cycle would in general be indeterminant—consider the analagous case of a set of one-way, non-delay switches arranged in a cycle).

(ii) the tree for any given module α never includes more than a finite number of modules (even if, for theoretical purposes, the group A is infinite).

The transition equation for X_α^t is:

$$X_\alpha^{t+1} = f\,(S'_{a_1(\alpha),1}(t),\ S'_{a_2(\alpha),2}(t),\ldots,\ S'_{a_k(\alpha),k}(t))$$

$$S'_{\beta,i}(t) = S_\beta^t\ ,\ \text{if } Y_{\beta i}^{t+1} = (\phi,\ldots,\phi)$$

$$\text{and } B_\beta^{t+1} \text{ not defined}$$

$$= B_\beta^{t+1}\ ,\ \text{if } Y_{\beta i}^{t+1} = (\phi,\ldots,\phi)$$

$$\text{and } B_\beta^{t+1} \in S$$

$$= f\,(S'_{Y_{\beta i1}(\beta),1}(t)\ ,\ldots,\ S'_{Y_{\beta ik}(\beta),k}(t))\ Y_\beta^{t+1},$$

$$\text{if } Y_{\beta i}^{t+1} \neq (\phi,\ldots,\phi)$$

$$\text{If } Y_{\beta ij}^{t+1} = \phi \text{ then define } S'_{\phi(\beta),\,j}$$

$$= S_{a_j(\beta)},\ \text{if } B_{a_j(\beta)}^{t+1} \text{ not defined}$$

$$= B_{a_j(\beta)}^{t+1},\ \text{otherwise.}$$

Under interpretation the transition equation for X_α^t specifies the storage state, X_α^{t+1}, in terms of the states at time t, S_β^t, of the modules β belonging to the connection tree for α. Note that, because of the recursive definition of $S'(t)$, f may be iterated several times in the determination X_α^{t+1}—compare this to the determination of the output of a tree of switches without delays.

IV

TOWARD A THEORY OF ADAPTIVE SYSTEMS

As already mentioned, the work reported here is part of a larger effort which has as its goal a theory of adaptive systems. The effort is an individual one and, of course, reflects particular biases of the author. This section will discuss the relation of the present paper to the broader program.

The first step of this program was the description of a computer which could simulate the operation and, in certain respects, the structure of any automaton (growing or fixed). The second step summarized here in part III, consisted in giving a general and formal description of computers like the one first obtained—the iterative circuit computers. The resulting mathematical characterization represents a broad class of machines, of arbitrary geometries, etc.; by an appropriate choice of $(A, A^\circ, X, \{P_i\}, f)$ it is possible to represent directly not only the behavior but also the changing structure and local operation of any given potentially-infinite automaton, tessellation automaton, n-tape Turing machine, or growing logical net. (In this respect note, for instance, that the class of iterative circuit computers properly contains A. Church's class of potentially-infinite automata—any two modules in an iterative circuit computer may eventually become connected so that either affects the other in a single time-step, whereas in a potentially-infinite automaton the corresponding delay, in time-steps, increases with increasing separation and is a constant for any given separation).[3]

For an iterative circuit computer the ideas of sub-program and automaton are, in an important sense, interchangeable. For any automaton, a sub-program can be written which not only has the same behavior but also the same changing structure and local operation. On the other hand, a given sub-program will in general occupy a finite number of modules in the computer and will have its action (or state) determined by bordering modules and the input function B. Thus, as a survey of the characterizing equations will show, an automaton or growing logical net can be constructed which mimics the sub-program.

It is important to note that sub-programs can be set up which, for instance, can shift themselves from one set of modules to another set, i.e. from one position to another. Thus the underlying geometry of the iterative circuit computer (given by A and A°) in effect determines the geometry of a space in which the sub-program is embedded; the transition equations then serve, in a sense, as the laws of this universe. For this reason the sub-programs of a given iterative circuit computer will often be spoken of as "embedded automata". This view of the results of the second step leads directly to the third step.

The central object of the third step is to provide formal apparatus for the <u>implicit</u> definition of automata—definition by means of generators and relations on these generators. Implicit definition of an automaton is analogous to the implicit definition of an algebraic group. In the case of the group, instead of giving an explicit listing of the elements of the group and their interrelations, the group is defined (often quite compactly) in terms of a set of generating elements and relations on products of the generators. In a similar sense an automaton can be implicitly specified by an initial set of elements and a set of growth rules.

The mathematical characterization of iterative circuit computers provides the apparatus needed for a precise formulation of automaton generators. Specifically, the generators will be sub-programs which can be thought of as (relatively) elementary embedded automata having the following properties:

(i) movement—the generators will in general be capable of shifting as a unit from one position to another (as specified by input $B(\alpha,t)$ or the state of adjacent modules—note that motion may be random if the input sequences $B(\alpha,t)$ are random),

(ii) connection—generators will combine under conditions specified internally (within the sub-programs) to form larger sub-programs capable of moving and acting as units,

(iii) production—generators can alter the state of adjacent modules (note that a sufficiently complicated generator could directly duplicate itself).

The generators will act upon other generators or other sub-programs present in the computer ("precursors") by connecting them or breaking them into components. The generators will all be acting simultaneously and if a given generator duplicates itself the duplicate will also in general be active.

So that any possible automaton can be defined in terms of the generators it is necessary to choose the set of generators so that any possible program for the computer can be represented by an appropriate connected set of generators (cf. the process of picking a set of instructions sufficient for a universal computer). Once this is done, the generation of particular automata can be effected by controlling rates of production and connection, movement and contact, the nature of precursors present, etc. That is, the relations on the generators will consist of specifying the initial states and input sequences which control such factors. For any given iterative circuit computer and set of generators, the relations possible can be given an appropriate equational form. Under one approach, the way in which the generators are initially connected and the nature of the precursors in the "environment" are sufficient together to specify the generated automaton. Things become particularly simple if the generators cannot interpenetrate when moving (a kind of "billiard ball physics").

A given program, because of the loops or iterations, is much more compact than the complete sequence of steps in the calculations it controls. In the same sense the connected system of generators which specify a given automaton will be much more compact than the automaton generated. Thus the automaton can have scattered throughout its structure complete implicit descriptions of its structure—such considerations play an important part in the study of self-repairing automata.

In the implicit definition of an automaton certain feedback phenomena play a crucial role. The feedback phenomena can be to some extent isolated by observing at what level a given generator system falls in the following succession of categories (each of which properly includes its successor):

(i) productive systems—the generator system produces other generators or precursors,

(ii) autocatalytic systems—the generator system produces generators or precursors which are used in its construction, i.e. the system produces some of its own components,

(iii) self-duplicating systems—the generator system produces duplicates of itself.

Such considerations lead directly from step three to step four and from work in progress to work which lies in the future.

The central object of step four will be to define the term "adaptive system" for embedded automata. Because of the formal nature of the definition, it then becomes possible to investigate these adaptive systems deductively (and by simulation). The beginnings of such a definition lie in the following consideration: With the help of concepts such as autocatalytic and self-duplicating generator systems it is possible to define such concepts as steady-state equilibria and homeostasis for embedded automata. In fact one can go quite far in this direction obtaining discrete state relaxation processes (Southwell), morphogen standing-wave phenomena (Turing) and so forth.[11,13] Automata exhibiting these properties will usually have the desirable property that small changes in structure result in small changes in behavior (at least over a certain range). Thus the behavior of a given automaton of this type gives some indication of the behavior of all automata of similar structure ("hill-climbing" techniques become applicable). If the generator system for such an automaton has a hierarchical structure, then a small change in structure produces a small effect in proportion to the "position" of the change in the hierarchy. That is, a generator system may consist of autocatalytic or homeostatic systems, systems of these (which may or may not be autocatalytic or homeostatic), etc.; changes at the upper levels of the hierarchy will generally have a greater effect than those at lower levels. By making changes first at the highest level and then at progressively lower levels of the hierarchy, it should be possible to narrow down rather quickly to any automaton in this category having some initially prescribed behavior.

Changes in the generated structure result when relations on the generators are altered. The effect of such alterations can perhaps be more clearly seen under the following interpretation. The generation of a particular automaton can be looked at as if all possible generation processes are going on simultaneously but at different rates. Some will be going very slowly (infinitely slowly in the limit) while others will be proceeding very rapidly. The <u>way</u> in which these rates are changed in order to change the generated automaton will have important consequences with respect to the adaptiveness of the overall system (cf. A. L. Samuel's work on changing the weights of a "checker-move tree").[10]

As a final point it should be noted that the environment of an embedded automaton can be made as simple or complex as desired. Since adaptation must be defined in terms of the range of environments in which the automaton is to be embedded, this is an important factor. It has already been noted that the environment may contain other sub-programs (precursors) or in fact other embedded automata. The latter case amounts to an implicit definition of the environment since only the initial state and internal rules of each of the embedded automata need be given. Contrast this with an explicit definition which would require a point-by-point, time-step-by-time-step description of the state of the environment. If the precursors in the environment are relatively elaborate and sophisticated then the adaptation process will look similar to a heuristic learning system (cf. Newell-Shaw-Simon).[8] If precursors are absent or simply generators then the adaptation process will look more like the processes considered by Friedberg.[4] It seems likely that implicit definition of the environment will play an important part in the development of step four.

BIBLIOGRAPHY

1. Burks, A. W., *Computation, Behavior, and Structure in Fixed and Growing Automata*, University of Michigan Technical Report ONR Contract 1224(21), 1959.

2. Burks, A. W., and Wang, H., "The Logic of Automata," *J. Assoc. Computing Mach.*, *4*, 193-218, 279-297 (1957).

3. Church, A., *Application of Recursive Arithmetic in the Theory of Computers and Automata,* notes from summer conference course in Advanced Theory of the Logical Design of Digital Computers, The University of Michigan, 1958.

4. Friedberg, R. M., "A Learning Machine: Part 1," *IBM Journal of Research and Development*, *2*, 2-13 (1958).

5. Holland, J. H., "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously," *Proc. 1959 Eastern Joint Computer Conference*.

6. Kleene, S. C., "Representation of Events in Nerve Nets and Finite Automata," in: *Automata Studies*, *Annals of Mathematics Studies*, no. 34, Princeton, 1956.

7. Moore, E. F., *Machine Models of Self-Reproduction*, Paper 560-52 at October Meeting of the American Mathematical Society, Cambridge, Mass. 1959.

8. Newell, A., Shaw, J. C. and Simon, H. A., *Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics*, Report P-951, Rand Corporation, 1957.

9. Rabin, M. O., and Scott, D., "Finite Automata and Their Decision Problems," *IBM Journal of Research and Development,* *3*, 114-125 (1959).

10. Samuel, A. L., "Some Studies in Machine Learning, Using the Game of Checkers," *IBM Journal of Research and Development*, *3*, 210-229 (1959).

11. Southwell, R. V., *Relaxation Methods in Engineering Science; a Treatise on Approximate Computation*, Clarendon Press, Oxford, 1940.

12. Turing, A. M., "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proc. Lond. Math. Soc.* (2), *43*, 230-265 (1936).

13. Turing, A. M., "The Chemical Basis of Morphogenesis," *Phil Trans. Roy. Soc.*, ser. B, *237*, 37 ff. (1952).

14. Von Neumann, J., *The Theory of Automata*, unpublished manuscript.

UNITED STATES ARMY SIGNAL RESEARCH AND DEVELOPMENT LABORATORY
DISTRIBUTION LIST
CONTRACT NO. DA36-039 SC-78057

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

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

Chief of Research and Development
OCS, Department of the Army
Washington 25, D. C.
Attn: Dr. I. R. Hershner, Jr.

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

Chief Signal Officer
Department of the Army
Washington 25, D. C.
Attn: SIGRD

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

Director
U S National Bureau of Standards
Boulder Laboratories
Boulder, Colorado
Attn: Div 14-0, Library

Commander
Wright Air Development Center
Wright Patterson Air Force Base
Ohio
Attn: WCOSI-3

Commander
Air Force Cambridge Research Center
L. G. Hanscom Field
Bedford, Mass.
Attn: CROTLR-2

Commander 2
Rome Air Development Center
Griffiss Air Force Base
New York
Attn: RCSST-3

Commander 10
Armed Services Tech Info Agency
Arlington Hall Station
Arlington 12, Va.

Massachusetts Institute of Technology
Lincoln Laboratory
Cambridge 39, Massachusetts
Attn: W. Davenport, Jr.

Director
National Bureau of Standards
Washington 25, D. C.
Attn: Dr. S. Alexander

Commanding Officer
Office of Ordnance Research
Box CM, Duke Station
Durham, N. C.

Commanding General
Ballistics Research Laboratory
Army Proving Ground, Maryland
Attn: Dr. C. V. L. Smith
Ch, Computation Lab.

Moore School of Electrical Engineering
University of Pennsylvania
220 South 33rd Street
Philadelphia 4, Pennsylvania
Attn: Prof. G. Patterson
Switching Theory Contract
Attn: Prof. S. Gorn

Army Mathematics Research Center 2
University of Wisconsin
Madison, Wisconsin
Attn: Dr. R. Langer

Commanding General 2
U S Army Signal School
Fort Monmouth, New Jersey

Commanding Officer
U S Army Signal R and D Laboratory
Fort Monmouth, New Jersey
Attn: NP (Dir Data Proc Fac Div) 2
Attn: NPT (Ch Data Transducer Br) 2
Attn: G (Mathematics Div.
Mr. L. Leskowitz, Computation Ctr.)
Attn: XS (Mr.J. Borsuk, Explor Res Div S)
Attn: XS (Dir Explor Res Div S)
Attn: N (Dir Systems Engrg Div)
Attn: ADTE (Evans)
Attn: NR (Dir Trans Fac Div)
Attn: RHA (Records Holding Area)
Attn: TN (Tech Info - For: 3
Brit and Can Covts)
Attn: DR (Ofc Dir of Res Opns) 5

Staff
Stanford Electronics Laboratories
Stanford University
Stanford, California

This Report is distributed by Office of Research Operations, Fort Monmouth, New Jersey
Telephone: Liberty 2-4000, Ext. 52335.