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

Quarterly Report No. 1
15 April 1962 - 15 July 1962

MACHINE ADAPTIVE SYSTEMS

Arthur W. Burks
John H. Holland

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

August 1962

TABLE OF CONTENTS

PROGRAMMING AND THE THEORY OF AUTOMATA

Arthur W. Burks

The second part of the last quarterly report was devoted to a general characterization of the late John von Neumann's ideas for a theory of automata. Von Neumann was the first to conceive of and propose the development of a general theory of automata which would explain the computational and control aspects of both natural information processing systems (e.g., nervous systems) and artificial information processing systems (e.g., digital computers).

At about the same time that von Neumann started work on his theory of automata Norbert Wiener proposed the development of a similar subject, which he called cybernetics. A few words of comparison of Wiener's cybernetics and von Neumann's theory of automata are in order, particularly because of the strong current interest in "bionics". Bionics is not a new subject as so many people seem to think, but only a branch of cybernetics and of automata theory.

There is a great deal of similarity of subject matter and interest between Wiener's cybernetics and von Neumann's automata theory, which is not surprising in view of the fact that there was a good deal of direct interchange between the two men. From our point of view the most important difference is that von Neumann's theory was more closely tied to large digital computers than Wiener's cybernetics; this is no doubt connected to the biographical fact that von Neumann participated much more heavily in logic and computers than did Wiener. Both von Neumann's automata theory and Wiener's cybernetics were highly programmatic: their main achievement was to show that certain known results in biology, biochemistry, logic, computer design, and computer use could be made the basis for a comprehensive theory about control, programming, information, and computer design. Hence both cybernetics and automata theory are important mainly as sources of problems, methods, and suggestions. Von Neumann did accomplish substantial results in two

areas. First, the study of probabilistic automata, particularly with respect to the problem of synthesizing reliable computers from unreliable components. Second, the design of self-reproducing automata.

Wiener presented his conception of cybernetics in many published works, and it is now very well known. Von Neumann published little on automata theory during his life, and all he did publish was fragmentary and tentative. In view of the importance of his ideas they deserve to be better known. The purpose of the second section of our Fourth Quarterly Progress Report was to organize them and present them so as to provide a jumping-off place for further work in this area.

Von Neumann anticipated that modern technology would lead to computers many orders of magnitude more complex than those built in the first decade after the war. Recent developments in cryogenics, thin films, and tunnel diodes show that the components for such computers are here, or at least near at hand. Whether the computer industry is ready to design and construct very large systems composed of such components is more problematic. Aside from the fabrication problems involved in large systems of small components there are two theoretical problems, the solutions to which would greatly aid in the construction of large and complicated computing systems: the reliability problem, and the problem of how to organize very large automata. Both of these problems belong to the theory of automata.

Von Neumann's position on this last point was even stronger than the one just stated: he felt that it would be impossible to build very complicated computers until the theory of automata was much further developed than at the time he wrote. Whether or not such an extreme position is correct, it seems to the writer and his colleagues in the Logic of Computers Group that developments in the theory of automata along the lines envisaged by von Neumann will in time have important applications to actual machines.

It has been recognized from the very beginning of the development of

general-purpose computers that a computer consists of both hardware and software, and that while some of each is needed a great deal of interchange from one to the other is possible. For example, division may be made a primitive operation of the computer, or it may be programmed from addition, multiplication, and a conditional control command. It is important that from a theoretical point of view the logical design of machines and the "logical design" of programs involve the very same principles. It is equally important that there is a programming and logical design aspect to natural self-reproduction.

In the present paper we seek to abstract the common element from (1) programming as it occurs in the theory of automata, including Turing machines (2) automatic programming systems for actual computers and (3) the programming and control aspects of self-reproduction. Because in connection with natural systems, biologists have practically no information on this third point, our information on this must come from automata theory (i.e., the work of von Neumann and work that has grown out of it). Biologists, biophysicists, and biochemists have learned a great deal in recent years about self-reproduction, both at the cellular and the non-cellular level, and including much about the informational aspects of self-reproduction, but they have not yet learned much about the over-all control and programming processes of self-reproduction. Perhaps the present decade will see important advances in this area.

We begin by reviewing A. M. Turing's universal simulation result.

"On Computable Numbers, with an Application to the Entscheidungsproblem." Proceedings of the London Mathematical Society, Series 2, 42 (1936-37) 230-265. "A correction," ibid., 43 (1937) 544-546. We will not follow Turing's formulation exactly.

Two different types of computing units are involved. First, a tape consisting of an indefinite number of squares, each capable of storing a single symbol. The tape is indefinitely expansible in one direction, but it is essential that

initially only a finite number of squares are "marked", i.e., are not blank. The second type of unit is a finite control automaton with a "tape head", i.e., a finite automaton which can scan and read a square of the tape, erase that square or write in it, and at the next step scan the square to the left or the right of the square just scanned. A Turing machine consists of a finite control automaton connected to a tape.

A finite control automaton and an indefinitely expansible tape are very different, the former being finite and active, the latter being potentially infinite and passive. But both are kinds of automata, constructible out of switch and delay elements and passing deterministically from one discrete state to another. The exact relations between them are of interest to the theory of automata and programming. We will study these relations first from the point of view of Turing machines and later on from the point of view of von Neumann's self-reproducing automaton.

Let us establish the following conventions. The addresses of the squares are 0, 1, 2, ... The finite control automaton of a Turing machine will scan initially the square which has zero as its address. There is a fixed alphabet for the tape, with characters of three types: a blank (the initial state of almost all squares), non-numerical characters, and numerical characters. The finite amount of information recorded on the tape initially (i.e., before the finite control automaton begins to function) will be expressed by alphabetic characters located in the consecutive even-numbered squares 0,2,4,..., 2n, where n is a non-negative integer. This information will be called the program and it will never be erased, i.e., we consider only finite control automata which do not alter any part of the program found initially on the tape. It is easy for the finite control automaton to sense the first blank even-numbered square and thereby sense the end of the program. The computed answer will be expressed in numerical characters printed consecutively in the even-numbered squares following those with the program. The odd-numbered squares will be used for

"scratch work." M*T is the Turing machine composed of the finite control automaton M and the tape T. It should be kept in mind that T is just a tape, not a "tape unit"; the circuits for shifting and altering T are in M.

It is clear that the answer computed by a Turing machine is a digital sequence, and that, under the fiction that there is a radix point (e.g., a binary point) to the left, the sequence represents a real number between zero and one. The case where the Turing machine produces no answer (i.e., a null sequence) may be accommodated by speaking of the "null number." Let $\eta(M*T)$ be the number computed by M*T.

Now let α be the class of numbers computed by all Turing machines, i.e.,

$$\alpha = \{x \mid (\exists M)(\exists T) \quad \eta(M*T) = x\}$$

We may think of α as being generated by $\eta(M*T)$ as the variable "M" varies over all finite control automata and the variable "T" varies over all tapes. But it is not necessary to vary both M and T at the same time to obtain α; α may be generated either by varying M while keeping T fixed or by varying T for a suitable fixed M.

The first of these results is easily established. Let Λ be the tape which is initially all blank and let β be the class of numbers computed by Turing machines with blank tapes:

$$\beta = \{x \mid (\exists M) \quad \eta(M*\Lambda) = x\} \quad .$$

We show that $\alpha=\beta$. Consider a particular number $\eta(M*T)$. By definition the program on T is finite; let T' be the finite segment of T containing this program and let T" be the balance of T. Now a finite piece of tape with a program written on it is itself a finite automaton or is very nearly so. At any rate we can easily design a finite control automaton M' which incorporates both M and T' and which uses T" as its tape. Then $\eta(M'*T") = \eta(M*T)$, and since T" is totally blank, $\eta(M'*T") = \eta(M'*\Lambda)$. Hence $\alpha=\beta$. Moreover, since a finite control automaton can sense the end of the program on a tape, it can also erase that program. It

follows that for each tape T, α may be generated by varying M over the class of finite control automata, i.e.,

$$\alpha = \{x \mid (\exists M) \quad \eta(M*T) = x\} \quad .$$

It is obvious that M and T do not play dual roles in the generation of α. For example, if M_0 is a finite automaton which does nothing to the tape, $\eta(M_0*T)$ is the "null number" for every T. This lack of duality is just what one would expect, for though a tape is an automaton it is a passive one. Turing showed, however, that there is a "universal" finite control automaton M_u which is sufficiently powerful to generate α as T is varied over all tapes. Let

$$\gamma = \{x \mid (\exists T) \quad \eta(M_u*T) = x\} \quad .$$

Turing's result is, then, $\alpha = \gamma$. He showed this by defining (hypothetically constructing) M_u so that it has this property: For each finite automaton M there is a tape $\mathcal{P}(M)$ such that

$$\eta(M_u*\mathcal{P}(M)) = \eta(M*\Lambda) \quad .$$

The way M_u works is of great interest since it involves the important notion of simulation. Each finite control automaton M has a finite number of states. Its passage from one of these states to the next is determined by the tape symbol scanned; this behavior may be expressed by a finite state table. Each state of the finite control automaton produces a certain effect on the tape; this information likewise can be expressed in a finite table. These two tables constitute the program on $\mathcal{P}(M)$. Consider now the behavior of $M*\Lambda$. At each moment of time M is in one of its states, and the tape Λ has a finite sequence of symbols written on it; hence the complete state of $M*\Lambda$ can be expressed by a finite sequence of symbols. $M_u*\mathcal{P}(M)$ simulates $M*\Lambda$ in the following way. M_u inspects the program on $\mathcal{P}(M)$, determines the initial complete state of $M*\Lambda$, and records this complete state on its own tape. M_u then iterates the following step indefinitely: by examining the program on $\mathcal{P}(M)$ and the last computed complete state of $M*\Lambda$ it computes the next complete state of $M*\Lambda$. As the digits of $\eta(M*\Lambda)$ are computed, $M_u*\mathcal{P}(M)$ records

these in the even-numbered squares beyond the program. Since M_u can simulate any machine M in this way, M_u is a universal simulator.

It is easy to see that there are infinitely many finite control automata M' which satisfy the defining property of M_u, that is, such that for each finite control automaton M there is a tape $\mathcal{Q}(M)$ for which

$$\eta(M' * \mathcal{Q}(M)) = \eta(M * \Lambda) \quad .$$

We will call any such automaton M' a universal control automaton. To recapitulate: all elements of α may be obtained with a blank tape, and all may be obtained with a single universal control automaton.

The procedure used earlier to prove $\alpha = \beta$ is the reverse of the procedure used later to prove $\alpha = \gamma$. In the former a program was transformed into an automaton, while in the latter an automaton was transformed into a program. The latter transformation is much more significant than the former, partly because it is more difficult to show that it works, but more fundamentally because of the difference between a finite control automaton and a tape. A finite control automaton is active and has the power to interpret a program, while a tape is passive and can do nothing by itself. A practical corollary of this difference is the fact that it is much more difficult to design a computer than it is to prepare a tape. We buy one computer and use it to solve a wide variety of problems by varying the program fed into it.

Let us explore further the practical bearings of Turing's universal simulation result. The first point to note concerns the infinitude of a computation. The tape of a Turing machine is potentially infinite and the computed output is in general an infinite sequence. In contrast, all actual computers and all actual computations are finite. This difference does not, however, really matter in the present instance, for we can restate the universal simulation result for finite computations. Let $\eta_f(M*T)$ refer to the first f digits of $\eta(M*T)$ if they exist, otherwise to $\eta(M*T)$. The universal control automaton M_u has this property: For

each finite automaton M there is a tape $\mathcal{Q}(M)$ such that for each f, $\eta_f(M_u * \mathcal{Q}(M)) = \eta_f(M*\mathcal{A})$.

Thus in the application we are making of Turing's universal simulation result the fact that a Turing machine is potentially infinite while an actual computation is finite does not matter. It should be noted that there are applications of Turing machine theory where this difference is essential. For example, people often attempt to use certain results about Turing machines to help answer the question "Is man a machine?" Such results as the non-existence of certain decision procedures

---

Turing, op. cit., shows that there is no decision procedure for whether $\eta(M*\mathcal{A})$ contains a particular symbol, or whether $\eta(M*\mathcal{A})$ is an infinite sequence.

---

and the fact that no Turing machine can enumerate all mathematical truths

---

Kurt Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I," Monatshefte für Mathematik und Physik 38 (1931) 173-198. Gödel showed that there is no complete axiomatic system of arithmetic, from which it follows that the set of mathematical truths cannot be enumerated by a Turing machine.

---

are cited as being relevant to the question. But if man is an automaton he is most certainly a finite automaton, so results about Turing machines, which are infinite, do not apply directly to the question "Is man a machine?" There may be indirect connections of importance to this question, but no one has yet shown that there are.

Our second point of comparison between a Turing machine and an actual computer concerns the distinction between a special-purpose and a general-purpose computer. It is natural to think of $M*\mathcal{A}$ as a special-purpose computer, since its function is to compute the single number $\eta(M*\mathcal{A})$, and to think of M_u as a general-purpose computer, since, suitably programmed, it can compute any number $\eta(M*\mathcal{A})$. But this way of

looking at things can easily be misleading. The most essential difference here is between a single Turing machine M*T and an infinite class of such machines obtained by varying M while keeping T fixed or by varying T while keeping M fixed. Moreover, as "special-purpose" and "general-purpose" are normally used by computer people they connote practical rather than theoretical concepts. Most so-called special-purpose machines are universal control automata in the sense defined earlier. That this is so is fairly evident when it is realized that there exists a universal control automaton with eight states operating on tapes having an alphabet of five symbols.

Shigeru Watanabe, "5-Symbol 8-State and 5-Symbol 6-State Universal Turing Machines," Journal of the Association for Computing Machinery 8 (October 1961) 476-483.

Moreover, any actual special-purpose computer is used to solve a number of problems, not just to compute one number $\eta(M*T)$, and hence is programmed in some sense. A computer is called general-purpose when it is relatively easy to program or use it on any of a wide variety of problems, and it is called special-purpose when in practice it can only be used to solve a relatively narrow class of problems. The distinction between special-purpose and general-purpose computers is thus a bifurcation of the class of universal control automata based on a practical criterion.

This brings us to our third, and last, point of comparison between a Turing machine and a general-purpose computer. We referred to the information placed initially on the tape T as a "program," but it might just as well have been called the "data," for we can think of the machine M as defining a function from T to $\eta(M*T)$. To make the usual distinction between program and data we must divide the information placed initially on the tape into two parts, one part to be called the program and the other part the data. We then think of the program as defining

a function to be computed by the machine, the data as constituting the argument or arguments of the function, and the computed output as being the function value for those arguments. As so described, the distinction between program and data is purely arbitrary, and this is certainly so from a purely formal point of view. For example, it is arbitrary whether we say a number referred to in a conditional shift of control (branch) command belongs to the program or the data or both, and when a program is the object of computation (e.g., in compiling) this program is the data. Our criterion for distinguishing program from data is an informal, intuitive one: the program is that which, for the most part, directs operations; the data are those items which, for the most part, are operated upon.

At this point it is desirable to use a slightly more complex notation for the tape. Let I be the finite part of T containing the input information. The rest of the tape is blank, and hence is Λ, so T may be represented by $I^\frown\Lambda$. In this notation, each finite control automaton M defines a function from I to $\eta(M^*I^\frown\Lambda)$. The behavior of the universal control automaton M_u may then be stated as: For each finite control automaton M there is a finite description $\mathcal{D}(M)$ such that

$$\eta(M_u^*\mathcal{D}(M)^\frown\Lambda) = \eta(M^*\Lambda)$$

In this new notation "$\mathcal{D}(M)$" refers to the finite portion of the tape holding the description of machine M, whereas earlier it referred to the whole tape which contained this description.

M_u is a universal simulator of Turing machines with blank tapes. Is there a universal simulator of Turing machines whose tapes are of the form $I^\frown\Lambda$? It follows from considerations similar to those given earlier that there is. By our earlier convention we considered only tapes in which the squares 0, 2, 4, ..., 2n, where n is a non-negative integer, were occupied by alphabetic characters (excluding the blank) and all other squares were blank. Let us now also consider tapes in which the squares 2n+4, 2n+6, ..., $2n_1$ may also be occupied by alphabetic characters, where $n_1 \geq n+2$. A tape $\mathcal{D}(M)^\frown I^\frown\Lambda$ will then consist of a description of machine M on squares 0, 2, 4, ..., 2n followed by the input information I on squares 2n+4,

$2n+6, \ldots, 2n_1$. A finite control automaton M together with a tape of the form $I^{\frown}\Lambda$ defines a function from I to $\eta(M^*I^{\frown}\Lambda)$. There is a finite control automaton M_u^1 which will simulate any such machine, that is, for each M there is a finite tape $\mathcal{D}(M)$ such that

$$\eta(M_u^1 * \mathcal{D}(M)^{\frown} I^{\frown}\Lambda) = \eta(M^*I^{\frown}\Lambda).$$

The basic principle and design of M_u^1 is the same as that of M_u.

The foregoing involved the extension of our earlier conventions so as to allow two distinguishable blocks of information on a tape. This extension can be carried on indefinitely, using the squares

$0, 2, 4, \ldots, 2n$

$2n+4, 2n+6, \ldots, 2n_1$

$2n_1+4, 2n_1+6, \ldots, 2n_2$

.

.

.

where $n_1 \geq n+2$, $n_2 \geq n_1+2$, etc. A universal simulator may be designed for each level, the universal simulator using one more block of input information than the machines it simulates. Thus M_u uses one block of input information to simulate all machines M with blank tapes:

$$\eta(M_u * \mathcal{D}(M)^{\frown}\Lambda) = \eta(M^*\Lambda) \quad ;$$

M_u^1 uses two blocks of information to simulate all machines M with one block of input information:

$$\eta(M_u^1 * \mathcal{D}(M)^{\frown} I^{\frown}\Lambda) = (M^*I^{\frown}\Lambda) \quad ;$$

M_u^2 uses three blocks to simulate all machines M with two blocks of information:

$$\eta(M_u^2 * \mathcal{D}(M)^{\frown} I^{\frown} I^{1\frown}\Lambda) = \eta(M^*I^{\frown} I^{1\frown}\Lambda) \quad ;$$

etc., etc., etc.

Automatic programming can be analyzed in these terms, and turns out to be a case of M_u^2 using three blocks of input information. Let M_m be a general-purpose

computer (finite automaton) produced by some manufacturer. In the present state of technology M_m will operate with a "machine language" inconvenient for the programmer to use. Let P be the program expressed in this machine language and D the data for a computation. Then a single "run" of the machine computes the finite number $\eta_f(M_m{*}P^\frown D^\frown \Lambda)$, and if the run were extended to infinity and the machine was supplied with an unlimited amount of tape it would compute $\eta(M_m{*}P^\frown D^\frown \Lambda)$. Suppose now you have constructed a "programmer's language"

Arthur W. Burks, "The Logic of Programming Electronic Digital Computers," Industrial Mathematics 1 (1950) 36-52. See p. 51.

(automatic programming language) in which a programmer can very easily write a program and express the input data of a problem. It is theoretically possible to build a machine which will understand this programmer's language directly; call this hypothetical programmer's computer M_p. Supplied with program P and data D this automaton will compute $\eta(M_p{*}P^\frown D^\frown \Lambda)$.

Since both M_p and M_m are universal control automata they are theoretically equivalent, their differences being practical. The hypothetical computer M_p was designed so as to match the human problem-giver well, that is, so as to be easy to program. But M_p is impracticable in the present state of technology, the manufacturer's machine M_m being the best machine actually available. Automatic programming seeks to bridge this gap by doing with "software" what cannot be done with "hardware," to use the current computer jargon. Automatic programming works in the following way. We write a coded description $\mathcal{D}(M_p)$ of the hypothetical machine M_p in the machine language of the manufacturer's computer M_m. In the usual terms $\mathcal{D}(M_p)$ is the interpretive routine which translates from the programmer's language to the machine language. Then M_m operating under the direction of $\mathcal{D}(M_p)$ will compute the same function as M_p. That is, for any program P and data D, written

in the machine language of M_p

$$\eta(M_m * \mathcal{Q}(M_p)^\frown P^\frown D^\frown \Lambda) = \eta(M_p * P^\frown D^\frown \Lambda) \quad .$$

Since P and D are written in the machine language of M_p the programmer need know nothing about the machine M_m on which the computation is carried out. He need only understand the machine language of the hypothetical machine M_p.

A comparison of the last two equations

$$\eta(M_u^2 * \mathcal{Q}(M)^\frown I^\frown I^1{}^\frown \Lambda) = \eta(M * I^\frown I^1{}^\frown \Lambda)$$

$$\eta(M_m * \mathcal{Q}(M_p)^\frown P^\frown D^\frown \Lambda) = \eta(M_p * P^\frown D^\frown \Lambda)$$

shows that an automatic programming system is an instance of Turing's universal simulation result, with M_m playing the role of M_u^2, M_p of M, P of I, and D of I^1. But not every such instance of Turing's universal simulation result is a case of automatic programming. What is the differentia? That is, what do we require of a universal control automaton M_u^2 and a translation routine $\mathcal{Q}(M)$ so that they constitute an automatic programming system. The additional condition is a practical, human-oriented one: automatic programming is an application of universal simulation which makes programming automatic for homo sapiens. Thus it is essential to the idea of automatic programming that the hypothetical programmer's computer M_p be easier for the programmer to use than the actually available manufacturer's computer M_m. In other words, the computation system $M_m * \mathcal{Q}(M_p)$ must be matched to the human much better than the system M_m alone.

Let us consider for a moment the role of language in the operation and theory of computers. From the point of view of current automata theory, automata are just devices which jump discretely from state to state under the direction of input symbols. Thus in proving his universal simulation result Turing shows how to express the states and state transitions of a finite automaton M in a sequence $\mathcal{Q}(M)$. This requires only a rudimentary language. No concept of a machine language with a complicated structure corresponding to the structure of a machine appears in automata theory. But such languages are needed in working with actual computers. The reason for this is obvious. The number of states assumed by any actual computer

in even a short time is much too large for a state-by-state analysis to be of much use in viewing the whole computer, though such a mode of analysis is often valuable in designing small portions of the computer. In planning and operating a computer we must, therefore, organize it into interrelated units or compounds at various levels: half-adders, flip-flops, etc. at the lowest level; registers, memory switches, counters, etc. at the next higher level; arithmetic units, memories, input-output units, control, etc. at a still higher level. The particular organizations of current computers reflect strongly the present state of technology and our current problem interests, and I feel sure that radically new organizations will appear in the future. But in any case we must organize computers into some such hierarchy of units simply because they are too complicated for us to understand otherwise.

The organization of the machine provides the basis for the structure of the machine language. Thus a typical machine language is based on commands, one part of which designates an arithmetic or control operation, the other part of which designates memory locations. Now the machine language was designed from the machine's point of view. This is as it should be, but because man and machine are so different the result is that the machine language is not well suited to the human. For that reason we construct a programmer's language whose structure is much closer to the structure of ordinary language. To do this is, as we noted earlier, tantamount to designing a hypothetical programmer's computer which is well-matched to the human user.

To recapitulate: The organization of a machine into a hierarchy of interconnected units is essential for the understanding, construction, and operation of a complex computer. But none of this organization plays a significant role in automata theory as it is usually conducted. For the most part, current automata theory makes no distinction between a highly organized computer, on the one hand, and any other possible machine which behaves in the same way. Compare, for example, an IBM 7090 with another hypothetical machine which produces the same results but

which has no discernible machine structure and no machine language. Most results in automata theory apply equally well and in the same way to these two machines, which differ radically with respect to organization. As a consequence of this lack of theory, the design and instruction of digital computers is an art, the art by which man controls the machine.

The late John von Neumann sought a theory of the organization of automata which would be based on "that body of experience which has grown up around the planning, evaluating, and coding of complicated logical and mathematical automata"

The Computer and the Brain, p. 2. See also his "The General and Logical Theory of Automata," pp. 1-31 of Cerebral Mechanisms in Behavior - The Hixon Symposium (edited by L. A. Jeffress).

---

and which would have applications in the design and programming of digital computers. He outlined the general nature of his proposed automata theory: its materials, some of its problems, what it would be like, and the form of its mathematics. He began a comparative study of artificial and natural automata. And he formulated and partially answered two fundamental questions of automata theory: How can reliable systems be constructed from unreliable components?

"Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," pp. 43-98 of Automata Studies (edited by C. E. Shannon and J. McCarthy).

---

and, What kind of logical organization is sufficient for an automaton to be able to reproduce itself? His discussion of the last question is particularly relevant to programming.

Von Neumann has two models of self-reproduction, a kinematic and a cellular one.

The first is described in "The General and Logical Theory of Automata," loc. cit. The second is described in a manuscript to be published by the University of Illinois Press under the editorship of the present writer. It is also described briefly in C. E. Shannon, "Von Neumann's Contributions to Automata Theory," pp. 123-129 of Bulletin of the American Mathematical Society, Vol. 64, No. 3, Part 2, May 1958.

The cellular one is better for our purpose since it abstracts from kinematic problems of motion and enables one to concentrate on organizational and structural problems.

The basis of the cellular model is an infinite array of square cells, each consisting of a finite automaton with 29 states. The state of a cell at time t+1 is a truth-function of its own state and that of its four contiguous neighbors at time t. There is a fiducial state U, called the unexcitable state, and it is stipulated that at time zero all but a finite number of cells are in state U. The unexcitable state U corresponds to the blank state of a square of the tape of a Turing machine. As noted earlier, initially all but a finite number of the squares of the tape of a Turing machine are blank. Thus a Turing machine is initially homogeneous except for a finite area on the tape and for the finite control automaton interacting with the tape. Similarly, the basic framework for a self-reproducing automaton is homogeneous, and at time zero all but a finite number of cells are in this homogeneous state.

There is not time to develop the details of the construction and destruction processes which may take place in this cellular system, but a few comments will suffice for our purposes. As a first approximation we can think of the signals transmitted through the system as being of two kinds: computation signals of the usual kind, and construction-destruction signals. Construction signals may be used to shift a cell from its unexcitable state into a particular switch-delay state. Destruction signals may also be used for a destructive shift of a cellular

finite automaton back into its unexcitable state. A finite complex of cells in a particular switch-delay configuration constitutes a complex finite automaton which can compute in the way an ordinary finite automaton computes. Its output signals may be converted into construction-destruction signals.

Consider now the finite area whose cells are initially in some state other than U. It will be a finite automaton; call it the "primary automaton." Consider next some other area (finite or infinite) of cells that are in state U. The primary automaton can send signals into this area so as to organize it into a "secondary automaton." At any time this secondary automaton is finite, but it can get larger from time to time. It is thus possible to have "special purpose" construction in this cellular system: a primary automaton constructs some particular secondary automaton.

A universal Turing machine can be embedded in the cellular system. A finite area is designed so as to be a finite control automaton. An infinite linear array of cells constitutes the tape. The finite control automaton organizes the cells immediately above and below the tape into a reading loop which passes through one cell (square) of the tape. The finite control automaton can extend and contract this loop by means of construction-destruction signals. Note that in the embedded Turing machine the tape and control both stand still, and reading and writing are done by means of an indefinitely expansible variable length loop, whereas according to our earlier description of a Turing machine the finite control automaton and the tape move relatively to each other. Structurally these two modes of operation are different, but computationally they are equivalent. All that is required in a Turing machine is that the finite control automaton have ultimate access to each square of the tape.

Von Neumann showed how to design a universal constructing automaton M_c in this system. It is synthesized out of the following three main parts. (I) A finite control automaton which can read any position of an infinite tape. This operates in

the same way as the finite control automaton of the universal Turing machine just described. The tape itself consists of an infinite linear array of cells. The tape is used to store the description $\mathcal{D}(M)$ of an arbitrary secondary automaton which is to be constructed by M_c. Each automaton M is finite and hence each $\mathcal{D}(M)$ is also finite, but there is no limit to the size of $\mathcal{D}(M)$. Since the tape is not bounded in size it is not part of M_c. (II) A finite automaton which can interpret an arbitrary description $\mathcal{D}(M)$. Parts (I) and (II) acting conjointly can read and interpret $\mathcal{D}(M)$ and send construction-destruction signals to a specified secondary area so as to construct M there. (III) A finite automaton which can reproduce the tape of (I) along with its contents and attach the results to the secondary automaton M. Parts (I), (II) and (III) operating together constitute the universal constructing automaton M_c.

The finite automaton M_c operates in the following way. When the initial tape contents of the universal constructing automaton M_c consist of a description $\mathcal{D}(M)$ of a finite machine M, the universal constructor will build M and copy $\mathcal{D}(M)$ onto the tape of M. This result may be written:

$$M_c * \mathcal{D}(M) \longrightarrow M * \mathcal{D}(M) \quad .$$

Self-reproduction is obtained by placing a description $\mathcal{D}(M_c)$ of the universal constructing automaton on its own tape. The construction formula just written applies to any finite automaton M, and since M_c is a finite automaton the formula applies to M_c. Substituting "M_c" for "M" in the formula we obtain:

$$M_c * \mathcal{D}(M_c) \longrightarrow M_c * \mathcal{D}(M_c)$$

which, in a logical sense, is a case of self-reproduction.

Let us pause for a moment to compare Turing's system with von Neumann's. On the surface they appear very different, but the fundamental logical and information theoretic principles on which they operate are very similar. Both are closed systems with a denumerable number of states, making deterministic transitions between states. Furthermore, both systems are composed of finite automata. Each square of a Turing machine tape is a two-state finite automaton. A square of tape

is not a very powerful automaton, to be sure, since it can do nothing by itself but can only interact with the finite control automaton, but it is nevertheless a finite automaton. Thus, initial states of the system aside, Turing's system consists of an infinite one-dimensional homogeneous array of finite automata with a single more complicated automaton attached to one element of the array. In contrast, von Neumann's system consists of an infinite two-dimensional homogeneous array of finite automata.

In each case the system is used by imposing a finite amount of inhomogeneity on it. This initial inhomogeneity can spread without bound throughout the system as time progresses. The infinitude of the system provides the matrix or background for unlimited growth, and the finite initial configuration of the system controls the pattern of this growth. Thus a Turing machine consists of a finite control automaton plus an unlimited amount of blank tape. Similarly, a von Neumann cellular automaton consists of a finite automaton plus an unlimited number of cells in state U which may be modified by that automaton.

With this comparison of a Turing machine and a von Neumann cellular system in mind let us look at Turing's universal simulation result once more. In the proof that $\alpha = \beta$ a program was transformed into an automaton, while in the proof that $\alpha = \gamma$ an automaton was transformed into a program. Thus programs and automata are sometimes interchangeable. When designing the ENIAC

This was the first electronic "general-purpose" digital computer. It is described in my paper "Electronic Computing Circuits of the ENIAC," Proceedings of the Institute of Radio Engineers 35 (August, 1947) 756-767.

---

we expressed this as a choice between constructing ("wiring in") an operation (such as division) and instructing (programming) that operation. Today the point would be made by saying that "hardware" and "software" are, within limits, interchangeable.

The "within limits" refers, of course, to the fact that there must be a finite control automaton which is active and has the power to interpret a program. Thus when a Turing machine is embedded in a von Neumann cellular system, the finite control automaton must extend and contract the reading loop by which the tape is read as well as direct the computation on the basis of what is recorded on the tape. A program is really a state of an automaton, so the interchangeability of programs and automata is really an interchangeability of state and system. One is reminded here of the analogy between kinetic energy and potential energy. A finite automaton M is active – a form of kinetic energy – while its description $\mathcal{D}(M)$ recorded in a succession of cells is passive – a form of potential energy. And just as passive programs and active automata are interchangeable within limits, so are potential and kinetic energy. In both cases some activity (kinetic energy) is required in the system initially unless it is to remain forever quiescent.

Von Neumann was interested in an existence result concerning the logic of self-reproduction: he sought a formal system with a reasonably minimal base in which one can construct an automaton that will reproduce itself in a manner logically similar to actual self-reproduction. Hence his choice of a relatively weak automaton as the occupant of each cell and his restriction that information cannot propagate any faster than one cell per time step. As a consequence, the construction of a self-reproducing automaton in his system is exceedingly involved and complicated. Moreover, practical considerations dictate that it operate serially, because parallel operation results in complicated, ad hoc phasing and interlocking problems.

Systems more directly relevant to computer design and programming and the study of adaptive and evolutionary processes can be obtained by strengthening von Neumann's basis. This is done principally by placing much more powerful finite automata in the cells, and secondarily by relaxing the restrictions on the speed of transmission of information between cells. The second modification also makes parallel operation feasible. Self-reproduction becomes simple in such a system,

See for example, my "Computation, Behavior and Structure in Fixed and Growing Automata," Behavioral Science 6 (January 1961) 5-22.

and can, of course, become trivial, as when there is a cell state 6 such that a cell in state 6 will cause any immediate neighbor in state U to go directly into state 6.

One type of modification of this form has been made by John Holland, who calls his systems iterative circuit computers. Another modification has been tentatively considered by myself, and I will outline it here.

The idea of an automatic programming language is a commonplace now and it is customary to teach this language to the user of a machine, rather than the machine language. As noted earlier, an automatic programming language is the machine language of a hypothetical programmer's machine M_p with a certain organization, and this organization is presupposed in the automatic programming language. This suggests that it would be better to teach the potential user of a machine about the hypothetical machine M_p in conjunction with its language rather than to teach the automatic programming language in isolation from this hypothetical machine.

But what I wish to propose goes further than this. The hypothetical machine M_p was designed to solve all problems of a very wide class, and hence does not take advantage of the special properties of a particular problem. This limitation is inherent in the idea of a general-purpose computer. For many problems it is easier to think of the problem in terms of a special-purpose computer especially designed to solve that problem. In doing this one will not be distorting his natural way of formulating a problem to adapt it to a particular computer. Instead,

he can formulate the algorithm for solving his problem by designing a special-purpose computer analogous to the problem.

I suggest, then, that instead of always writing a program for a problem one should sometimes design a special-purpose computer for that problem. No doubt this suggestion seems preposterous. But the moral to be drawn from the work of Turing and von Neumann is that programs and computers are, to a large extent, interchangeable. Since this is so there cannot really be such a great difference between writing a (special-purpose) program and designing a special-purpose machine as it seems at first sight. There appears to be a great chasm between these two types of activities because the comparison between machine design and program writing is usually drawn between the long, expensive, involved design procedures which have produced our present general-purpose computers, and the relative ease of writing a program in a given rigorously formulated program language. But this contrast is not the relevant one here. The engineering design of an actual computer involves much more than the purely logical design of the computer, and this purely logical design is constrained by these engineering considerations. Moreover, in writing out or diagramming the logical design of a computer one does not have available a rigorously formulated design language comparable in power to the best current automatic programming languages.

Hence my proposal involves the development of a framework or language of great expressive power for specifying the logical structure of any computer. Experience in machine design and the use of flow charts for programming suggests that this language be diagrammatic as well as symbolic. Moreover, it is feasible to build a computer which can scan a two-dimensional diagram, so that the design of a machine in this language can be fed directly into the manufacturer's machine M_m. In other words, in designing a machine M one is writing $\mathcal{D}(M)$ in the proposed machine design language. The machine M_m must be instructed how to interpret the expressions written in the machine design language – this calls for an interpretive routine $\mathcal{I}$. To

summarize: when one is interested in a computation $\eta(M*D\frown\lambda)$ he writes $\mathcal{Q}(M)$ and gives it to the machine-program complex $M_m*\mathcal{I}$. The number $\eta(M_m*\mathcal{I}\frown\mathcal{Q}(M)\frown D\frown\lambda)$, which equals $\eta(M*D\frown\lambda)$, is then produced.

Thus, my proposal involves, first, the development of a rigorously formulated machine design language, and second, the development of a routine $\mathcal{I}$ for the automatic translation of expressions in that language into the machine language of the actual machine M_m. These two steps are, of course, the same as those required for the development of an automatic programming system $M_m*\mathcal{Q}(M_p)$: the machine language corresponding to the programmer's machine M_p must be worked out and the interpretive routine $\mathcal{Q}(M_p)$ must be written. Likewise, the use of the automatic system $M_m*\mathcal{I}$ is similar to the use of the automatic programming system $M_m*\mathcal{Q}(M_p)$. In both cases one is given a problem. To solve the problem on $M_m*\mathcal{I}$ he writes a description $\mathcal{Q}(M)$ of a machine M which is equivalent to that problem. To solve the problem by means of $M_m*\mathcal{Q}(M_p)$ he writes a program P which is equivalent to that problem.

The systems $M_m*\mathcal{I}$ and $M_m*\mathcal{Q}(M_p)$ operate on different levels of the hierarchy of Turing machines introduced earlier. It will be recalled that the universal machine M_u uses one block of input information to simulate Turing machines with blank tapes, M_u^1 uses two blocks to simulate machines with one block of input information, M_u^2 uses three blocks to simulate machines with two blocks, etc., etc. In this hierarchy $M_m*\mathcal{I}$ is a case of M_u^1, as is shown by the formulas

$$\eta(M_m*\mathcal{I}\frown\mathcal{Q}(M)\frown D\frown\lambda) = \eta(M*D\frown\lambda)$$

$$\eta(M_u^1*\mathcal{Q}(M)\frown I\frown\lambda) = \eta(M*I\frown\lambda) \quad ,$$

and M_m used with $\mathcal{Q}(M_p)$ is a case of M_u^2, as is shown by the formulas

$$\eta(M_m*\mathcal{Q}(M_p)\frown P\frown D\frown\lambda) = \eta(M_p*P\frown D\frown\lambda)$$

$$\eta(M_u^2*\mathcal{Q}(M)\frown I\frown I^1\frown\lambda) = \eta(M*I\frown I^1\frown\lambda) \quad .$$

There are many possible approaches to our proposed machine design language. We will briefly indicate an approach which is suggested by von Neumann's cellular self-reproducing automaton but which diverges from it in a number of important

respects. A finite or growing automaton of any power may be stipulated as the contents of a cell, provided that the specification of the automaton, either directly or via a chain of definitions, is reasonably simple. Thus one cell could store a number, with the understanding that the cell can store as many (finite) digits as the number has. For example, if it stores a ten bit number $\underline{x}$ to begin with and is to store $\underline{x}^2$, $\underline{x}^3$, $\underline{x}^4$, ... at various stages during the computation, the cell will automatically grow in size so as to accommodate the extra bits that are produced by successive mulitplications.

In specifying a problem by means of a special-purpose computer one would assume as many serial stores, parallel memories, control units, etc., as was convenient. Data could be organized into blocks in natural ways. The control automata stipulated could direct operations like: sum the series in block A, monotonize the data in blocks B and C, withdraw from memory all sequences having property Ø, etc. There would be provision in the machine design language for defining new automata in terms of old ones, so once an automaton is specified others can easily be designed in terms of it.

Von Neumann has a fixed crystalline structure for his cells. We propose to allow new cells to spring up between old ones under the control of the computation. Suppose a list of words is stored in bins and at a later date new entries are to be inserted. This change would be conceived as an automatic process of inserting new storage bins between the old ones. This change must, of course, be accompanied by an appropriate change of the switches which connect these bins to the rest of the automaton. In general, storage and computing facilities would be created wherever needed and in a form suited to the problem being solved. Hence a batch of information would not be stored in a homogeneous memory, as is the case in current computers, but in a memory organized so as to reflect the organization of the information itself. That is, the memory would be divided into categories, sub-categories, etc., in natural and useful ways, cross-switching connections would be assumed where needed, etc.

Current computers are organized into large, specialized units such as memories, arithmetic units, and controls. The reasons for this organization are to be found in the nature of the components from which computers are built. Since the special-purpose computers to be designed in our proposed machine design language are not to be built, there is no reason for organizing them in the conventional way. Rather, they should be organized in whatever way best accommodates the problem at hand. Consider, for example, a two-dimensional partial differential equation. It may be convenient to solve this equation by computing the value of a function at all grid points simultaneously, in which case the special-purpose computer should be organized so as to do this. It should be clear from the foregoing that in our proposed machine design language one could formulate machine organizations radically different from present ones.

In conclusion, let us review briefly how one would use the proposed machine design language. It would be most effective when applied to a problem capable of analog treatment, i.e., whose structure may be paralleled by the structure of a special-purpose computer which will solve the problem. In such a case the mathematical equation describes the behavior of a physical model. To specify the solution of this equation one describes in the machine design language a special-purpose computer which would operate analogously to the given physical model. The description of this special-purpose computer is supplied to a general-purpose computer which translates it into its own machine language and then solves the problem.

I believe that the system just proposed for instructing and using digital computers is practically feasible, though admittedly it would require a great deal of development work. In any case, I think that the theoretical possibilities of it, together with Turing's and von Neumann's results on universal machines, illuminates the general nature of digital computers and the problem of their relation to man, the machine-user.

CONCERNING EFFICIENT ADAPTIVE SYSTEMS

John H. Holland

1. Introduction

This study of adaptive efficiency is based upon the framework described in "Outline for a Logical Theory of Adaptive Systems" [5]. Its principal object is to give some idea of the methods appropriate to that framework. Two considerations prompted the choice of "adaptive efficiency" for this purpose:

1) the study of efficiency has lead to important results in related areas such as information theory;

2) preliminary work indicated that, along this line, statistical mechanics applies in a direct and simple way to the generation procedures and generated systems of automata theory.

The results are derived in a narrow and elementary context, but the methods used (and the results) appear to extend naturally to broader contexts.

Section 2 ('General discussion') gives a general statement of the observations motivating the paper. The translation of these observations to the more rigorous framework provided by automata theory requires first of all an appropriate class of automata; the class of automata used has been described elsewhere [6] and only a few relevant points will be reviewed in section 3 ('Abstract basis'). Section 4 ('Adaptive efficiency') considers the concept of efficiency as it applies to a restricted subclass of the systems described in section 3. It is shown that acquisition of new information generally forces a concurrent, less efficient use of information already accumulated. The main result gives the rate of acquisition of information which (as a function of the concurrent loss of efficiency) produces optimal adaptation. Loosely, the result shows both that no system of the type considered should consign more than half of its effort to acquisition of new information, and that there are

reasonable conditions under which this limit should be closely approached. The final section ('Summary') discusses the particular results and their generalization to the full range of systems described in section 3.

2. General discussion

Classically, adaptation is a process whereby an organism is modified to fit it (or its progeny) more perfectly for existence in its environment. This classical view, if one pays it close heed, sets distinctive conditions on a theory of adaptive systems: The statement emphasizes that it is not sufficient to characterize just the system's internal processes. The environment (or range of possible environments), the information received therefrom and the ways the system can affect the environment must also be characterized. And, when this is done, the phrase "...to fit it more perfectly for existence..." still requires attention. The phrase suggests both a condition and a way of meeting it: Clearly, the theory should provide a formal means of determining which of two organizations is the fittest in a given environment. That is, the theory should include a fitness-measuring function, either directly or indirectly defined, enabling comparision of systems and selection of the fittest. Only then is it possible to discuss the process of adaptation within the context of the theory. "...to fit it...for existence..." suggests that the fitness measure be defined indirectly in terms of a survival criterion. As a matter of fact the problem of fitness can quite generally be rephrased in terms of survival under conditions imposed by the environment [4]. The result is a differential selection of systems according to fitness — the fittest systems at any point in time being those which have persisted. In general terms, then, the core of a theory of adaptive systems should be a study of differential selection.

A living organism constantly exchanges material and energy with its environment — an environment which, even when it is homogeneous in the large, involves important local variations. (One can observe similar interactions in any adaptive system, natural or artificial.) If a species of organism is to survive it must at least partially control this interaction. Because the environment does vary, this control can only be apropos if the organism receives and employs information about its environment. A system receiving no information from its environment obviously cannot base its control procedures upon the condition of the environment; "complete" information, on the other hand, opens the possibility of perfect control. Moreover, moving from the first extreme of no information to the second of complete information, one would expect a steady improvement in control possibilities. If the information is received at a rate r, then adaptation is limited accordingly; a system can alter its organization no more rapidly than it receives the relevant information. Thus, the efficiency of a system's control is limited by the rate at which it receives relevant information from the environment. And, of two similar systems in similar environments, the one exercising more efficient control will be the one favored by differential selection. Or so it seems intuitively.

It is another matter to give precision to this argument. The argument as presented has, in common with most heuristic arguments, a very loose texture. It can hardly be validated or invalidated in its given form and it is almost impossible to ascertain what elements of truth lie within it. In particular, although a relation between adaptive efficiency and information input rate is suggested, the precise nature of the relation and its usefulness can only be evaluated in a more rigorous context. The remainder of the paper will be devoted to a deeper look at this relation.

3. Abstract basis

This paper makes contact with automata theory through the formally defined class of iterative circuit computers. As the introduction notes, an earlier paper [5] describes the use of this class as a basis for studies of adaptation. Only points relevant to the present inquiry will be touched upon in this section.

In general terms, the study of adaptation is a study of how systems generate methods enabling them to adjust efficiently to their environments. Let us equate "method" with "program of a specified universal computer." From a more abstract viewpoint, then, an adaptive system is a schema for generating programs in accordance with the dictates of the environment. Let any such schema be called a generation procedure. The population of programs generated at a given time can be considered the repertory of methods available to the adaptive system at that time. The discussion here will be based upon generation procedures defined in terms of a selected finite set of programs, called generators, and a graph, called a generation tree:

1) No restriction is to be placed upon the programs selected for the set of generators; in one case the set of generators may include just the equivalents of individual instructions, in another it may include highly sophisticated heuristic programs. If some generation procedure is supposed to generate all programs of a specified universal computer, the set of generators must include a complete set of instructions for that computer.

2) The generation tree specifies the combining processes whereby the programs are formed from the generators. The vertices of the tree can be divided into two categories: major vertices and auxiliary vertices. Each major vertex of the graph represents a distinct program which can be formed

from the generators. Each auxiliary vertex indicates a distinct combination process — it can be thought of as labelled with a parameter indicating the rate at which the process is taking place. Given the generation tree and an initial population of programs it is possible to determine the population of programs to be expected at any later time. A change in the rate of any combination process produces a change in the generated population sequence. That is, a change of any rate constitutes a change of generation procedure.

With this refinement, adaptation becomes a matter of modifying the parameters associated with the auxiliary vertices of the generation tree — the object being to change the generation procedure in such a way as to produce combinations of programs better suited to the environment. Assume that the set of generation procedures accessible to the adaptive system has been specified. Then the process of adaptation induces an orbit (in the set-theoretic sense) upon the set. Different systems of adaptation will induce different orbits. If the different orbits can be assigned ratings then this abstract basis can be used to formulate questions of adaptive efficiency.

The problem of assigning ratings to orbits will be discussed near the end of this section; there is, however, a prior problem: realization of generation procedures. As in information theory, realizations are of considerable importance — the channel capacity theorems gain much of their significance from the companion "realizability" theorems which state the existence of codes permitting transmission rates arbitrarily close to capacity (i.e. efficiencies arbitrarily close to 1). Models of the foregoing generation procedures can be constructed along the following lines:

The generators can be thought of as embedded in a discrete or cellular space, each type occuring with a given density (the expected number of generators in some fixed number of cells). Each generator undergoes a random walk

in the space. Upon coming into contact with another generator it may, with a probability determined by generators involved, connect to it (connected sets of generators correspond to programs). At the same time combinations of generators may, with a probability again determined by the types of generators involved, separate into component combinations. The rate at which generators come into contact (the generation rate) together with the connection and disconnection probabilities determine the density of each type of program as a function of time and the initial densities. That is, these factors determine what programs are generated and in what order. Each choice of a set of connection and disconnection probabilities selects a particular generation procedure. Special programs — so-called templates — can be added to the model to modify the probabilities, a given population of templates thus determining a unique generation procedure. Let the generation procedure associated with a given template-free model be called a free generation procedure, and let the result of adding some templates to the given model be called a modified generation procedure. If the free generation procedure is taken as a basis, then each modified generation procedure produces a population of programs skewed relative to that of the free procedure. If it is assumed that there is a "cost" involved in producing templates, then we can associate a "skewing cost" (per unit time) with each modified generation procedure.

The iterative circuit computers mentioned earlier, will be used to provide formal realizations of these cellular spaces, their laws and processes. The iterative circuit computer is not in itself an adaptive system; it should be identified, rather, with the space in which the adaptive system is embedded. The generators and their combinations will be represented by programs in the computer. Because the iterative circuit computers have been characterized mathematically, they can be used as a formal basis for theorems about the embedded systems.

Each computer in the class is constructed of a single basic module (a fixed logical network) iterated to form a regular array of modules. An appropriate choice of the basic module will yield an iterative circuit computer structurally and behaviorally equivalent to any of the following types of automata: Turing machines (with 1 or more tapes) [9] [8], tesselation automata [10] [7], growing logical nets [1] [2], and potentially-infinite automata [3]. Programs of an iterative circuit computer can be given the following properties relevant to their interpretation as generators:

1) A given iterative circuit computer can execute arbitrarily many sub-programs simultaneously (within limits imposed by size).

2) Sub-programs can be written so that, under local control, they shift themselves from one set of modules to another set. Thus the geometry of the underlying iterative circuit computer becomes the geometry of the space in which the embedded program moves.

3) Sub-programs which are independent initially can move into contact (occupy adjacent sets of modules) and connect so as to form a larger sub-program capable of moving and acting as a unit.

4) Given any iterative circuit computer it is possible to select a finite set of sub-programs (individual instructions in the limiting case) to serve as generators such that any sub-program possible for that computer can be achieved by an appropriate combination of copies of these sub-programs.

5) By a suitable choice of the basic module it can be arranged that generators do not interpenetrate (or "over-write") when shifting (cf. the notion of a "billiard ball" physics).

Using the foregoing properties of generators one can realize any of the generation procedures described earlier in this section.

The question of rating orbits in the space of generation procedures can be treated now on the basis of the preceding discussion of realization. A given adaptive system can only be rated in terms of its performance in given environments. Thus some thought must be given first to characterization of permissible environments. It will only be noted here that an environment can be thought of as a population of problems presented to the system and that problems can be embedded in the same space as the generated programs. (That is, problems can be coded into the storage registers of the iterative circuit computer). On their random walk through the space the generated programs will encounter and attempt to solve the embedded problems. In other words, the population of generated programs acts upon the population of problems to produce solutions.

Let each problem type be assigned a numerical quantity called "activation" (the quantity might also have been called "reward" or "utility of solution"). This quantity is to be consigned or "released" to the adaptive system whenever the associated generation procedure solves the problem by means of one of its programs. Let A(G,E,t) be the expected net rate of activation release at time t when the generation procedure G is faced with environment E. (The net rate can be taken as the expected release rate minus the skewing cost associated with G.) Two generation procedures confronted by the same environment over an interval of time, T, can then be compared in terms of the expected net activation release over that interval. One can assign ratings in a similar way to orbits in the space of generation procedures.

Briefly then the problem of adaptation, in the framework outlined, becomes one of modifying a free generation procedure in order to maximize activation release from the environment. In the paper referred to earlier [5]

the rate at which a given adaptive system accumulates activation determines its survival (through duplication) in a population of adaptive systems. As a result the population of adaptive systems undergoes a differential selection according to ability to solve problems in the given environment. That system is fittest which in the long run accumulates the most activation.

4. Adaptive efficiency

4.1 Conditions, definitions, and initial considerations

The process of adaptation, as interpreted in section 3, involves a generated population of problem-solving programs acting upon a population of problems in an attempt to produce solutions. Assume that a set of generators, complete with respect to some universal computer, has been chosen and that the set of all possible programs formed from these generators has been enumerated and labelled $d_1, d_2, d_3, \ldots, d_i, \ldots$. Assume, furthermore, that a population of problems has been given and that it includes a denumerable number of different problem types $e_1, e_2, \ldots, e_j, \ldots$.

Following the description of section 3, let the programs and problems undergo random walks in the embedding space. We will concentrate our attention upon a bounded region of the space. Let the expected number of programs of type d_i in a unit volume at time t be a defined quantity, uniform over the bounded region (i.e. the programs are to be thought of as distributed with "uniform density"). Denote this quantity by $\rho(d_i)$; it will usually be referred to as the "density of d_i." Problems will be treated similarly and $\rho(e_j)$ will denote the "density of problems of type e_j." Let $\rho_d = \sum_i \rho(d_i)$, the "total program density," and $\rho_e = \sum_j \rho(e_j)$, the "total problem density," both be finite quantities.

Let $$\psi = \left[\frac{\rho(d_1)}{\rho_d}, \frac{\rho(d_2)}{\rho_d}, \ldots, \frac{\rho(d_i)}{\rho_d}, \ldots\right]$$

and $$\Theta = \left[\frac{\rho(e_1)}{\rho_e}, \frac{\rho(e_2)}{\rho_e}, \ldots, \frac{\rho(e_j)}{\rho_e}, \ldots\right] \quad .$$

ψ and Θ, as defined, are elements of Hilbert space. In what follows the problem population Θ is held constant while the program population ψ varies during the course of its adaptation to Θ (ψ thus being a function of time).

For the purposes of this example, let each problem be so designed that only one program at a time can attempt its solution. A problem will be called "unoccupied" during any interval when no program is attempting its solution; at all other times it will be designated as "occupied." An individual program, during its random walk, will encounter various ones of the embedded problems. It will be assumed that each time the program encounters an unoccupied problem it attempts a solution. Such an encounter, between a program of type d_i and an unoccupied problem of type e_j, will be called an attempt of type (i,j). Each attempt of type (i,j) is to last for an expected time t_{ij}, at the end of which the expected activation release is a_{ij}. That is, the program d_i "attempts to solve" the problem e_j for t_{ij} units of time (on the average) and is "rewarded" by an amount a_{ij} (on the average) for its efforts. If d_i does not even partially solve e_j then $a_{ij} = 0$. The a_{ij} become progressively greater for program types which provide more and more extensive partial solutions, reaching a maximum for programs (if any) which solve the problem completely.

Because each attempt at a solution involves an "occupation time" t_{ij}, a law of diminishing returns applies to attempts. As $\rho(d_i)$ is increased, for any i, an increasing proportion of problems of any type e_j can be expected to be occupied by the d_i. Thus as $\rho(d_i)$ increases, an individual of type d_i will find fewer and

fewer e_j unoccupied and, for that individual, the expected number of attempts of type (i,j) per unit time will decrease. But then, since activation release occurs only after an attempt, the expected rate of activation release per individual decreases also. In most cases of interest, the skewing cost per individual of a given type ultimately increases as one attempts to skew the generation procedure more and more to the generation of that one type. Under such conditions the net rate of activation release per unit volume for individuals of a given type will become negative as $\rho(d_i)$ approaches ρ_d. These conditions can be subsumed under the following general statement: A situation will be called competitive if, for any finite set of solvers D, the net rate of activation release per solver becomes negative as $\rho(D)$ approaches ρ_d.

It is important to note that, under competitive conditions, the discovery of each new $a_{ij} > 0$ is important to the adaptive system. For instance assume that at time t the system knows $a_{i_1j_1} > 0$ and has just discovered $a_{i_2j_2} = a_{i_1j_1}$; assume further that d_{i_1} cannot solve e_{j_2} and rejects it immediately so that $a_{i_1j_2} = 0$ and $t_{i_1j_2} = 0$ and, similarly, $a_{i_2j_1} = 0$, $t_{i_2j_1} = 0$. Let $\rho(d_{i_1}) = \rho_0$ just before the discovery of $a_{i_2j_2}$. Let $\rho(d_{i_1}) = \rho(d_{i_2}) = \frac{\rho_0}{2}$ after the discovery. Then other things being equal, a given program of type d_{i_1} (or d_{i_2}) will encounter an unoccupied problem of type e_{j_1} (or e_{j_2}) more frequently after the discovery than before. The increased attempt rate, coupled with the same total density of programs d_{i_1} and d_{i_2} and the fact that $a_{i_1j_1} = a_{i_2j_2}$, yields an increased expected activation release. (A similar advantage accrues when it is found that a single program can solve more than one problem.) Briefly, each new source, $a_{i_j} > 0$, allows reduction of the saturation effect by allowing the successful programs to be "spread" over a larger number of sources.

In the idealized situation so far presented, complete information about the environment would consist of complete knowledge of the matrix a_{ij}, the matrix t_{ij}, and the population Θ. The adaptive system possessing this information could then skew its program population ψ to maximize net activation release. Lacking such information, the system can acquire it only through encounters between elements of its program population ψ and the problem population Θ. Obviously the system can only use information already accumulated as a basis for skewing ψ. For example, consider a bounded adaptive system which initially has no information about its environment. As time passes encounters between its generated programs and the problems will provide information about some of the a_{ij} (and the associated t_{ij}). After a time t, if this information is accumulated, some finite subset, D(t), of program-types will be <u>known</u> to effect a positive release of activation from one or more problem types. The system can then modify its generation procedure accordingly.

One strategy the adaptive system can employ in adapting to the environment is to adjust ψ at time t so that <u>net</u> activation release from <u>known</u> values of a_{ij} is a maximum. That is, the density of elements of D(t) can be skewed so that activation release expected from programs belonging to D(t), <u>minus</u> the cost of skewing ψ, is a maximum. Under competitive conditions ρ_D, the total density of programs belonging to D(t), will be less than ρ_d.

We can show that this strategy is a "minimax" procedure for handling the environment. To see this, let a loss function $\lambda(\psi,\Theta)$ be defined in the following terms:

$$A(\psi,\Theta) \overset{df.}{=} \text{the expected net activation release (per unit time-volume) when program population } \psi \text{ operates on problems population } \Theta$$

$$\mu(\Theta) \overset{df.}{=} \psi_\Theta \text{ such that } A(\psi_\Theta,\Theta) = \max_{\psi} A(\psi,\Theta)$$

$$\lambda(\psi,\Theta) \overset{df.}{=} A(\mu(\Theta),\Theta) - A(\psi,\Theta)$$

In other words, the loss assigned to ψ in the presence of Θ is the difference between the maximum net activation release possible and that actually obtained by ψ.

$\{\Theta\}_t \overset{df.}{=}$ the set of problem populations having a_{ij} compatible with the values of a_{ij} known at time t.

$\psi_t \overset{df.}{=}$ the skewed program population resulting from the strategy of maximizing net activation release from the known a_{ij}.

$A_D(\psi,\Theta) \overset{df.}{=}$ the expected net activation release from program types belonging to the subset D when ψ operates on Θ (where D is some subset of the set of all program types, $\{d_i\}$)

Consider some $\psi \neq \psi_t$. By definition of ψ_t

$$A_{D(t)}(\psi,\Theta) \leqq A_{D(t)}(\psi_t,\Theta)$$

For any ψ, D(t), and $\epsilon > 0$ it is always possible to choose $\Theta' \in \{\Theta\}_t$ such that

$$A(\psi,\Theta') - A_{D(t)}(\psi,\Theta') < t$$

(Consider those problems $e_{j'}$ such that, for all i either $a_{ij'} = 0$ or else $a_{ij'}$ is unknown; choose $\Theta' \in \{\Theta\}_t$ such that the $e_{j'}$ are in fact solvable only by programs which occur with very low density in ψ)

But then $\min\limits_{\{\Theta\}_t} A(\psi,\Theta) \leqq \min\limits_{\{\Theta\}_t} A(\psi_t,\Theta)$ for any ψ

(Actually "min" should be replaced by "glb" in some cases.)

Therefore $\max\limits_{\{\Theta\}_t} [A(\mu(\Theta),\Theta) - A(\psi,\Theta)]$

$$\geqq \max_{\{\Theta\}_t} [A(\mu(\Theta),\Theta) - A(\psi_t, \Theta)]$$

Or $\max\limits_{\{\Theta\}_t} \lambda(\psi,\Theta) \geqq \max\limits_{\{\Theta\}_t} \lambda(\psi_t,\Theta)$ for any ψ

Or $\min\limits_{\psi} \max\limits_{\{\Theta\}_t} \lambda(\psi,\Theta) = \lambda(\psi_t,\Theta)$

Thus ψ_t "minimaxes" the loss function.

Recall that ψ_t is actually the program population which results if the adaptive system takes a very short term or opportunistic view of the environment — attempting to maximize net activation release from known sources without taking into account unknown sources. On the other hand, a minimax procedure is by its very nature a very conservative procedure based upon the most pessimistic estimates of the future. That the same population, ψ_t, results whether the adaptive system operates on a very opportunistic basis or upon a very conservative basis was at least unexpected. Nevertheless the adaptive system can do better than ψ_t.

4.2 Rate of information acquisition vs. limiting rate of adaptation, for homogeneous systems

The fact that ψ_t results from a minimax strategy in itself suggests that there are more efficient adaptive procedures — generally, a performance much better than minimax can be achieved at some small risk (often the strategy can be designed to make the risk as small as desired while obtaining significant improvement). As mentioned in section 3, two adaptive systems in a given environment are to be compared (over some interval of time T) in terms of their expected accumulations of activation. Thus our search is for a strategy which, over an interval of time T, will accumulate more activation than the strategy yielding ψ_t.

The possibility of improvement stems from the adaptive system's ability to control the rate at which it receives information from the environment. During any interval, say from t to $t+\Delta t$, the system can be expected to attempt some program-problem combinations (d_i, e_j) for the first time. These first attempts permit the system to add to its list of known a_{ij}. The system can control the expected number of first attempts in a given interval by controlling ψ. If ψ is changed so that the density of program d_i is increased, then the expected number

of encounters between d_i and various problems, e_j, will be increased and, as a result, the number of first attempts involving d_i will also increase. In effect, the first attempts constitute a random sampling of the environment biased according to ψ. Note that if both ψ and Θ are fixed, the expected number of first attempts steadily decreases; to maintain a fixed (expected) rate of first attempts the adaptive system must continually modify ψ.

Let any encounter resulting in a first attempt be called an informative encounter and let all other encounters be called non-informative.

$c(t) \overset{df.}{=}$ the expected number of encounters in the interval $(t, t+\Delta t)$

$\gamma(t) \overset{df.}{=}$ the proportion of encounters in the interval $(t,t+\Delta t)$ expected to be informative encounters.

By suitably controlling the population ψ the adaptive system can control γ and hence the rate at which it acquires new information about the environment. In what follows it will be assumed that the average informative encounter itself contributes negligibly to the expected activation release — negligibly because of the inefficiency of the exploration process in comparison with the process aimed at releasing activation from known sources. However the population ψ, while maintaining γ, can be modified to take advantage of the new information. That is, at any given time there are many populations ψ which will yield the proportion γ; certain of these populations use the newly discovered a_{ij} to better advantage than the population based upon the old list of known a_{ij}. By so modifying ψ in terms of the new information, the expected activation release, averaged over all non-informative encounters, will be incremented.

$\alpha(t) \overset{df.}{=}$ the expected activation release per non-informative encounter at time t.

$\delta(t) \overset{d.f.}{=}$ the <u>average</u> increment in $\alpha(t)$ to be expected from use of information from a single informative encounter at time t.

In terms of the above functions:

$$\alpha(t) = \alpha(0) + \int_0^t \delta\gamma c\,dt$$

If $\eta(t)$ is the skewing cost at time t, then the net activation release (per unit time-volume) at time t, A(t), is given by:

$$A(t) = \alpha(1-\gamma)c - \eta$$

$$\text{Let } A^*(T) = \int_0^T A(t)\,dt$$

$$= \int_0^T [\alpha(1- \ )c - \eta]\,dt$$

In general, of two homogeneous bounded adaptive systems of the same size, the one producing the greater $A^*(T)$ will have a selective advantage. The function γ is the control function which the system adjusts in an attempt to optimize $A^*(T)$ for the environment Θ. To gain some idea of how γ affects $A^*(T)$ let us impose the following simplifications:

Let Θ and hence c be constant. Constrain the adaptive system to operate with γ set to a constant $\bar{\gamma}$. Assume an average $\delta(t) = \bar{\delta}$ can be defined for $\delta(t)$ when $\gamma(t) = \bar{\gamma}$ and let $\eta(t) = k\bar{\gamma}c$ (the greater the number of informative encounters per unit time, the greater the skewing cost). Under these conditions

$$\alpha(t) = \alpha(0) + \int_0^t \bar{\delta}\bar{\gamma}c\,dt = \alpha(0) + \bar{\delta}\bar{\gamma}ct$$

and

$$A^*(T) = \int_0^T [(\alpha(0) + \bar{\delta}\bar{\gamma}ct)(1-\bar{\gamma})c - k\bar{\gamma}c]\,dt$$

$$= \alpha(0)(1-\bar{\gamma})cT + \delta\gamma(1-\gamma)c^2\frac{T^2}{2} - k\bar{\gamma}cT \quad .$$

The constant $\bar{\gamma}$ which maximizes $A^*(T)$ can now be determined; for a maximum

$$\frac{dA^*}{d\bar{\gamma}} = 0$$

or $$-\alpha(0)cT + \bar{\delta}c^2\frac{T^2}{2} - \overline{\delta\gamma}c^2T^2 - kcT = 0$$

whence $$\bar{\gamma}_0 = \left[\left(\frac{1}{2} - \frac{k+\alpha(0)}{\bar{\delta}ct}\right), 0\right]_{max}$$

For many interesting systems, T is determined by the time the system requires to accumulate some fixed amount of activation A_0^*. Upon accumulating A_0^* the system duplicates and both it and its offspring begin anew to accumulate A_0^*. If each of two systems requires A_0^* to duplicate, then the one with the smaller T will have a selective advantage. If, in addition, each system has the same small probability $p>0$, over an interval Δt, of disintegrating into its component parts, then the one with the smaller T will eventually displace the other completely in a mixed population of the two systems. γ_0, of course, yields the smallest possible T under the constraints imposed upon its calculation.

If $\bar{\delta}cT$ (the increment in α to be expected over time T if all encounters were informative) is large compared to $k+\alpha(0)$ (the skewing cost per informative encounter added to activation released initially per non-informative encounter) then $\gamma_0 \tilde{=} \frac{1}{2}$. Thus, under such conditions, the adaptive system's effort should be divided equally between obtaining new information and making use of information already available. If such a result were to hold under more general conditions, say under quasi-economic conditions, it would certainly be provocative — no economic entity spends anywhere near this much of its effort in research.

Attention should be drawn to the fact that this result was obtained under the assumption that the population ψ could be instantaneously modified to whatever form, under the constraint $\bar{\gamma}$, best suited the current set of known a_{ij}. Thus the result is a limiting result. If the time to adjust ψ is not negligible the performance will be degraded accordingly.

It should also be noted that the result was obtained under the assumption that the bounded adaptive system was homogeneous. If non-homogeneous systems are employed improvements can be effected. In particular one can employ boundaries (connected sets of generators) which selectively pass certain problem types. In this way the local density of given problem types can be increased within the bounded region. This makes possible an increase in the efficiency (attempt rate) of any program which solves one or more of the concentrated problems. (A more detailed discussion of non-homogeneous systems, including some results indicating the conditions under which improvement can be expected, will be published in another paper. One such result indicates the desirability of a high surface-to-volume ratio for the bounded regions.)

5. Summary

At first sight one might assume that an adaptive system can do no better than to use discovered sources of positive utility (activation, reward) as fully as possible from the moment of discovery onward. Such a strategy is a "minimax" strategy for the restricted class of systems investigated in this paper and, as it turns out, it is not in general the best strategy available to such systems. A fortiori, it fails as a "universal" strategy for adaptation.

The crucial step in determining a more efficient strategy comes when one tries to ascertain the net value, to the system, of additional information. For the systems considered, the gross value of additional information is set by the increment in activation release per operation which can be expected from the use of the information. (Such a valuation can be used for any system which by its operations can accrue positive utility from its environment.) In a competitive situation (where the availability of unoccupied sources is a function of program

density) each new source discovered can contribute to the increment in activation release per operation. Thus, under competitive conditions, information about a new source, or information about a single program which handles several known sources, or information about a program which handles a known source more rapidly, all has a non-zero gross value. To determine the net value of the information its cost must also be taken into account. The cost arises because the system must make less than full use of known sources in order to obtain additional information; the loss thus incurred is a cost which must be made up via use of the additional information. For the systems studied this cost arises when the generated program population is diverted from utilization of known sources (non-informative contacts) so as to increase the frequency of new program-problem combinations (informative contacts). The extent of this diversion is measured by the ratio, γ, of informative contacts to total contacts. The cost of maintaining γ at a given level is not a one-shot cost — if the generated population of programs remains constant, the number of informative contacts per unit time falls off exponentially. Under the conditions stated, each setting of γ corresponds to an acceleration of the rate of accumulation of activation. Now, one adaptive system is more efficient than another, in a given environment, if it can be expected to accumulate some critical amount of activation in a shorter time than the other. In systems for which γ is defined, the efficiency of the system, and its advantage under differential selection, thus depends upon the setting of γ (optimum values of γ have been determined for an elementary class of bounded homogeneous systems). On a broader scale, when gross value and cost are figured on the basis given, the net value of information is directly related to the selective advantage that information (used optimally) will give to the system.

Arguments similar to those used in the body of the paper can also be applied in more general cases. For example, if the problems, e_j, are stochastic processes

which the programs, d_i, are to predict, the argument goes through much as before except that the frequency of various program-problem combinations becomes important because the activation release associated with each combination is a random variable which must be estimated. The results also extend naturally to certain non-homogeneous systems, systems which are partitioned into subsystems by means of selective boundaries permitting passage of some programs and problems but not others. Such considerations lead directly to the study of coupled generation procedures and the study of the capabilities of systems realizing such procedures. It can be shown that non-homogeneous systems will in general have a selective advantage over homogeneous systems.

The results presented are, in themselves, quite elementary, although they may be suggestive to those inclined to view them as signposts; the various relations and definitions, though exhibited in this elementary context, have a broader scope. The primary purpose of the paper, however, has been to give evidence that it is possible to obtain results pertinent to adaptation within a framework provided by automata theory, even when only elementary methods are employed. It does not seem unreasonable that more sophisticated methods applied along similar lines will extend the results both in breadth and depth.

* * *

Acknowledgement

I would like to thank the members of the Logic of Computers Group at The University of Michigan for several helpful discussions of parts of this paper. The work it represents has been supported by the United States Army Signal Corps through contracts DA-36-039-sc-87174 and DA-36-039-sc-89085.

Bibliography

1. Burks, A. W., "Computation, Behavior and Structure in Fixed and Growing Automata," Self-Organizing Systems, Pergamon Press (1960).

2. Burks, A. W., and Hao Wang, "The Logic of Automata," J. Assoc. Computing Machinery, 4, 2 and 3, 193-218, 279-297 (1957).

3. Church, A., "Application of Recursive Arithmetic to the Problem of Circuit Synthesis," Summer Inst. Symb. Logic Summaries, 1957, Institute for Defense Analysis (1957).

4. Fisher, R. A., The Genetical Theory of Natural Selection, Dover, (1958).

5. Holland, J. H., "Outline for a Logical Theory of Adaptive Systems," J. Assoc. Computing Machinery, 9, 3 (1962).

6. Holland, J. H., "Iterative Circuit Computers," Proc. 1960 Western Joint Computer Conf., 259-265 (1960).

7. Moore, E. F., "Machine Models of Self-Reproduction," Proc. Sympos. Math. Prob. Biol. Sci., New York (1961).

8. Rabin, M. O., and D. Scott, "Finite Automata and Their Decision Problems," IBM J. Res. and Dev. 3, 3, 210-229 (1959).

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

10. von Neumann, J., The Theory of Automata: Construction, Reproduction, Homogeneity. (unpublished manuscript).