Outline for a Logical Theory of Adaptive Systems*

JOHN H. HOLLAND

University of Michigan, Ann Arbor, Michigan

"Und ein Lehrbuch des Glasperlenspiels soll dieser unser Aufsatz ja noch weniger sein, ein solches wird auch niemals geschrieben werden."

—Hesse

1. *Introduction*

The purpose of this paper is to outline a theory of automata appropriate to the properties, requirements and questions of adaptation. The conditions that such a theory should satisfy come from not one but several fields: It should be possible to formulate, at least in an abstract version, some of the key hypotheses and problems from relevant parts of biology, particularly the areas concerned with molecular control and neurophysiology. The work in theoretical genetics initiated by R. A. Fisher [5] and Sewall Wright [24] should find a natural place in the theory. At the same time the rigorous methods of automata theory should be brought to bear (particularly those parts concerned with growing automata [1, 2, 3, 7, 8, 12, 15, 18, 23]). Finally the theory should include among its models abstract counterparts of artificial adaptive systems currently being studied, systems such as Newell-Shaw-Simon's "General Problem Solver" [13], Selfridge's "Pandemonium" [17], von Neumann's self-reproducing automata [22] and Turing's morphogenetic systems [19, 20].

The theory outlined here (which is intended as *a* theory and not *the* theory) is presented in four main parts. Section 2 discusses the study of adaptation via generation procedures and generated populations. Section 3 defines a continuum of generation procedures realizable in a reasonably direct fashion. Section 4 discusses the realization of generation procedures as populations of interacting programs in an iterative circuit computer. Section 5 discusses the process of adaptation in the context of the earlier sections. The paper concludes with a discussion of the nature of the theorems of this theory.

Before entering upon the detailed discussion, one general feature of the theory should be noted. The interpretations or models of the theory divide into two broad categories: "complete" models and "incomplete" models. The "complete" models comprise the artificial systems—systems with properties and specifications completely delimited at the outset (cf. the rules of a game). One set of "complete" models for the theory consists of various programmed parallel computers. The "incomplete" models encompass natural systems. Any natural system involves an unlimited number of factors and, inevitably, the theory can handle only a selected few of these. Because there will always be variables which do not have explicit counterparts in the theory, the derived statements must be approximate relative to natural systems. For this reason it helps greatly that

* Received March, 1961; revised November, 1961.

the statements can be verified in terms of specific programming schemes for parallel computers; in this way the "complete" models provide confirmation and corroboration of the theory. At the same time one can discover for each "complete" model a set of analogous "incomplete" models. Thus the results pertaining to a given "complete" model, when suitably interpreted, also apply to the analogous "incomplete" models.

2. *General Plan*

The study of adaptation involves the study of both the adaptive system and its environment. In general terms, it is a study of how systems can generate procedures enabling them to adjust efficiently to their environments. If adaptability is not to be arbitrarily restricted at the outset, the adapting system must be able to generate any method or procedure capable of an effective definition. The intuitive idea of an effectively defined procedure has several equivalent characterizations, here, following Turing, the set of all effectively defined procedures will be identified with the set of all programs of some suitably specified universal computer. In these terms, unrestricted adaptability (assuming nothing is known of the environment) requires that the adaptive system be able initially to generate any of the programs of some universal computer. The process of adaptation can then be viewed as a modification of the generation process as information about the environment accumulates. This suggests that adaptive systems be studied in terms of associated classes of generation procedures—the associated class in each case being the repertory of the adaptive system.

There is an important change in viewpoint when one speaks of adaptation in terms of generation procedures. It is no longer a question of producing this or that particular program, rather it is a question of the ability to generate some particular population of programs. That is, with each generation procedure we associate the population of programs it generates; successive modifications of the generation procedure by the adaptive system correspond to successive alterations in the population of programs. It quickly becomes apparent that there is no reason to restrict the generation procedure to producing one program at a time. There is in fact a gain in generality if the generation procedure operates in parallel fashion, producing sets or populations of programs at each moment rather than individuals. (All this provided we can find a means of realizing such procedures, a question we shall tackle further on.) In the same vein we can treat the environment as a population of problems and, in fact, it soon becomes apparent that we can replace individuals by populations everywhere in the theory. By so doing we gain both in generality and in facility, opening the way to the application of statistics as well as logic.

A generation procedure which eventually produces any arbitrarily chosen program of some universal computer will henceforth by called a *universal generation procedure*. As noted earlier, it is sometimes essential that the class of generation procedures under study include universal generation procedures. To make subsequent discussion more concrete, a class of procedures containing an infinite

subclass of universal generation procedures will be described briefly here (in the next section the same class will be more precisely defined):

Each procedure will be defined in terms of a distinguished finite set of programs, called generators, and a graph, called a generation tree. 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 case it may include highly sophisticated heuristic programs The set of generators chosen will depend in part upon what phase of adaptation is being investigated—primitive sets of generators being indicated, for instance, when the study concerns the origin of complexity, sophisticated sets when problem-solving is the primary aim. If the procedure is to be a universal generation procedure, the set of generators must be complete with respect to some universal computer. That is, by combining copies of the generators it must be possible to construct any program of the underlying universal computer. The combining processes, whereby the programs are formed from the generators, are specified by the generation tree. The generation tree will be presented here only in terms of a descriptive model (it will be discussed in detail in the next section): The generators can be thought of as embedded in a discrete or cellular space, each type occurring 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.

It is important to note that a change in the connection probabilities alters the population of programs to be expected at any given time. In particular, assume that the generation rate and disconnection probabilities have been fixed. Then each choice of a set of connection probabilities selects a particular generation procedure from the class of admissible procedures. In short, control of the connection probabilities amounts to control of the generation procedure. We will see (section 5, Models) that special programs for modifying connection probabilities—so-called *templates*—can be added to the models described above. Let the generation process associated with a given template-free model be called a *free generation procedure*, and let the result of adding some set of 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" with each modified generation procedure.

The generated population of programs will act upon a population of problems (the environment) in an attempt to produce solutions. For adaptation to take place the adaptive system must at least be able to compare generation procedures as to their efficiency in producing solutions. In order to provide this comparison it will be assumed that to every problem is assigned a numerical quantity called "activation". ("Activation" is the name used in section 5; the quantity might also have been called "reward".) This quantity is consigned or "released" to the adaptive system whenever the associated generation procedure solves the problem by means of one of its programs. Let $r_G(E)$ be the average rate of activation release (assuming this quantity defined) when the generation procedure G is faced with environment E. Let c_G be the skewing cost per unit time (assuming

the template population to be in dynamic equilibrium) associated with G. The net rate of activation release, $r_G(E) - c_G$, can be assigned to G as its rating relative to E. (Other rating procedures are of course possible; see sections 5 and 6). Two generation procedures confronted by the same environment can be compared in terms of their ratings. Adaptation within this context involves two concurrent processes: (1) sampling of the environment in order to produce estimates of the environment; (2) modification of the generation procedure to obtain the generation procedure with highest rating relative to the current estimate.

It still remains to implement the conditions just set down. In terms of the example so far discussed a natural form of implementation consists of providing each adaptive system with a central control over its associated generation procedure. Central control can be achieved by introducing a process for producing templates while constraining each generation procedure to occupy a finite region in the model. (Boundedness allows definition of local populations and evaluation of their effectiveness.) By providing a special set of generators, the process for producing templates can also be presented as a program, called a *supervisory program* in section 5. The supervisory program serves as a device to gather the threads of control at a single point. It produces a given distribution of templates over time and the templates in turn give rise to a modified generation procedure. In this sense the supervisory program serves as an implicit definition of the distribution of problem-solving programs to be expected at any time t—to any given supervisory program will correspond a specific sequence (in time) of distributions (over type) of problem-solving programs. Changes in the supervisory program will, of course, result in changes in the population of problem-solvers.

Adaptation, then, is based upon differential selection of supervisory programs. That is, the more "successful" a supervisory program, in terms of the ability of its problem-solving programs to produce solutions, the more predominant it is to become (in numbers) in a population of supervisory programs. In order to implement differential selection two provisions are necessary: (1) supervisory programs must be capable of self-duplication in order to provide successive generations as grist for the selection principle; (2) the rate of duplication of a supervisory program must be determined by the effectiveness of the problem-solving programs it controls. The two provisions are satisfied if we assume, as with the templates, that there is a cost associated with the duplication of supervisory programs and that this cost is met by released activation. Then the rate of duplication of a supervisory program will be determined by the net rate of activation release of the associated local problem-solving population. Differential selection follows since (assuming spontaneous disconnection and fixed density of generators—see section 5) the program with the higher rate of duplication becomes ever more predominant in the population of supervisory programs. Of course a single supervisory program, even should it deviate widely from the "norm", would affect the rates of connection only locally. Thus useless deviations would initially have little effect upon the overall level of adaptation and (be-

cause of spontaneous disconnection) would shortly disappear from the population. On the other hand, if the deviation should prove effective, then an ever larger proportion of supervisory programs would incorporate it. In effect, "successful" local variations in the modification of the generation procedure would be propagated throughout the discrete space of the model.

Operation of the selection principle depends upon continued generation of new varieties of supervisory programs. There exist several interesting possibilities for producing this variation. Here we only note that it is possible to subject each position (generator) in the supervisory program to modification during duplication. More precisely, if there are k generator types $(g_1, \cdots, g_k)$, we can associate with the ith position of the supervisory program the vector $(p_{i1}, \cdots, p_{ik})$ where $p_{i1} + \cdots + p_{ik} = 1$ and p_{ij} is the probability that the jth generator will occur at position i. Thus we can associate an ordered set of probability vectors, i.e. a probability matrix, with each supervisory program. Under this condition the successive generations (duplications) of a given supervisory program will give rise to a population made up of a variety of supervisory programs. In the absence of selection, the distribution of types to be expected will be determined by the probability matrix associated with the initially given supervisory program. If a change is made in the set of initially given probability vectors, changes in the distribution of problem-solving programs can be expected. Our next concern will be the nature of these changes.

First, let us examine the successive generations (duplications) of two supervisory programs which differ only slightly in their corresponding probability matrices. It can be shown, in the absence of differential selection, that at first the two populations of supervisory programs arising from the initially given supervisory programs have similar expected distributions. This in turn means that the resulting distributions of problem-solving programs will be similar. In other words supervisory programs which are similar or resemble each other in terms of probability matrices can be expected to produce distributions of problem-solving programs which for a while differ only slightly. But then the level of adaptation, in terms of solutions produced, must be closely related in the two cases. Thus supervisory programs which are similar under the above measure will have related levels of adaptation in any given environment. This observation has several important consequences and the remainder of this section will be devoted to its ramifications.

Before going further note that the criterion of similarity mentioned above—a criterion which is relevant to the effects produced by the supervisory program—can be given a precise formulation. Consider the set of all distinct supervisory programs and the related collection of all probability matrices. In the collection of matrices we can define a distance function or metric as one of the usual measures of difference defined on matrices. Then, given two supervisory programs, the *distance* between them will be taken to be the difference of the associated sets of probability matrices. With the help of this metric, neighborhoods *in the set of supervisory programs* can be defined and discussed. As we consider neighborhoods of smaller and smaller diameter about a given supervisory program

the effects of other supervisory programs in the neighborhood will progressively approach that of the given program. This relation can be stated in another way. Let a random sample be drawn from some neighborhood in the collection of supervisory programs. Then, for neighborhoods of progressively smaller radius, the sample will give an increasingly good estimate of the adaptation of the other programs in the neighborhood.

When the factors producing differential selection are added to these models, the idea of sampling takes on additional significance. Under differential selection, the success of a supervisory program in effect biases the probability of trying further samples in that neighborhood. In greater detail: Duplication with modification, acting on an initially given population of supervisory programs, provides a continual supply of new variants. The degree of similarity to be expected between any given variant and its parent will depend upon the probability matrix, which itself will be subject to selection. New variants may occur as if they were drawn at random from the collection of supervisory programs; or, for example, probability matrices may predominate which make the likelihood of a given variant proportional to its similarity to the parent program. (In section 5 we will discuss conditions under which the latter arrangement would have a definite selectional advantage). If a supervisory program is relatively successful it will duplicate, producing a population which contains not only the original program but modifications of it. If the likelihood of a variant is proportional to its similarity to the original, many of the modified supervisory programs will be similar to the original program. If one of these modified supervisory programs develops a more successful population of problem-solving programs it will begin to predominate in the population of supervisory programs. The result will be that new modifications will increasingly come from the new supervisory program. In other words, under differential selection, the sequence of supervisory programs will "tune in" on the best supervisory program in the neighborhood of similar programs. On the other hand, not only will an unsuccessful supervisory program disappear from the population (because it does not duplicate or does so inefficiently) but, as a result, the chance of similar programs arising from it by modification will be eliminated. In this way not only is the program itself selected against, but in effect so is its entire neighborhood. The combination of the "tuning in" effect with this "avoidance" effect produces the sampling bias described at the beginning of the paragraph.

This completes the discussion of the general plan of the study—the next three sections will consider some of the more difficult points in greater detail.

3. *A. Class of Generation Procedures*

The preceding section introduced, as an example, a class of generation procedures containing an infinite subclass of universal generation procedures. There the class was discussed informally via a description of one of its models; here the class will be precisely defined in preparation for a more detailed discussion of its models in the next section. This particular class has been chosen because it is

relatively easy to define and yet extensive enough to be used to illustrate the main points of a study of adaptation based upon generation procedures.

As mentioned earlier each generation procedure is defined in terms of a set of generators and a graph called a generation tree. Let $(g_1, \cdots, g_k)$ be the set of generators. Each permissible combination of generators (each program) is represented by a vertex in the generation tree. The vertices are arranged in levels so that the vertices of the first level correspond to the individual generators (one vertex for each generator), the vertices of the second level correspond to all permissible combinations of two generators, etc. More precisely the generation tree can be defined as follows: Let $V(i, j)$ be the connected set of i generators corresponding to the jth vertex at level i, v_{ij} (where $V(1, j) = g_j$). Let H_k be the set of all pairs $(V(i', j'), V(i'', j''))$ such that $i' + i'' = k$. Let L_k be the collection of all *distinct* connected sets of generators which result from a single connection between the paired elements of H_k. Note that in general there will be several ways of combining a given pair; also different pairs may yield the same program after combination. (To keep things simple programs are constrained to combine only pairwise; three or more programs cannot be combined in a single operation.) To each element of L_k assign a single vertex, v_{kh}, at level k. For each pair in H_k which gave rise to $V(k, h)$ in L_k let there be a directed edge of the graph terminating in v_{kh}; each such edge will be joined, via an auxiliary vertex v_{kh}^r, to the vertices $v_{i'j'}$ and $v_{i''j''}$ corresponding to the pair of elements combined, $(V(i', j'), V(i'', j''))$. Note that the auxiliary vertex in the resulting "Y" configuration does not belong to a level; it will be used to represent the specific connection process which yields $V(k, h)$ from $V(i', j')$ and $V(i'', j'')$. Each auxiliary vertex v_{kh}^r will be labeled with two numbers, p_{kh}^r and q_{kh}^r, $0 \leqq p_{kh}^r \leqq 1$, $0 \leqq q_{kh}^r \leqq 1$, called the *connection* and *disconnection probabilities* respectively. It will be required for each main vertex v_{kh} that $\sum_r p_{kh}^r \leqq 1$ and $\sum_r q_{kh}^r \leqq 1$. Each main vertex v_{ij} will be labeled with a variable $d(i, j, t)$ designated *density*. In addition a single number, c, the *generation rate*, is associated with the graph as a whole.

The connection probabilities, the disconnection probabilities, and the generation rate are constants of the process; under interpretation the densities specify the number of programs of each type in a selected volume of the embedding space (the space in which the generation procedure is embedded). It remains to specify the transition equations giving the densities at time $t + 1$, $\{d(i, j, t + 1)\}$, as a function of the constants of the process and the densities at time t. The transition equations, for any generation procedure of the class, can be presented in the following form:

A vertex v' will be termed an *input vertex* (*output vertex*) of vertex v if there is a directed edge of the graph from (to) v' to (from) v. In particular, an auxiliary vertex will be termed an input (output) auxiliary vertex of vertex v if there is an edge from (to) the auxiliary vertex to (from) v.

Let $C(t)$, $t = 0, 1, 2, \cdots$, be a sequence of random variables having a common distribution for which the mean is defined and equal to the generation rate, c.

Similarly, let $X_u(t)$ be a random variable taking as values the triples (i, j, h) with probability

$$2p^h_{ij} \frac{d(i', j', t-1) \cdot d(i'', j'', t-1)}{(d_0(t-1))^2}$$

where $v_{i'j'}$ and $v_{i''j''}$ are input vertices of the auxiliary vertex v^h_{ij} which in turn is an input axuiliary vertex of

$$v_{ij}\ ; \quad d_0(t-1) \overset{df}{=} \sum_i \sum_j d(i, j, t-1),$$

the total density of programs at $t-1$.

Let $M_{ij} \overset{df.}{=} \{h \mid v^h_{ij}$ belongs to the set of input auxiliary vertices of $v_{ij}\}$.

Define $\delta_{ijh}(i', j', h') = \begin{cases} 1 & \text{if } (i', j', h') = (i, j, h) \\ 0 & \text{otherwise.} \end{cases}$

$e_1(i, j, t) \overset{df.}{=} \sum_{u=1}^{C(t)} \sum_{M_{ij}} \delta_{ijh}(X_u(t)) \overset{df}{=}$ production of $V(i, j)$ through connection at time t.

Let $Y_u(i, j, t)$ be a random variable taking as values the triples (i, j, h) with probability q^h_{ij} and the symbol ϕ with probability $1 - \sum_h q^h_{ij}$.

$e_2(i, j, t) \overset{df.}{=} -\sum_{u=1}^{d(i,j,t-1)} \sum_{M_{ij}} \delta_{ijh}(Y_u(i, j, t)) \overset{df.}{=}$ loss of $V(i, j)$ through disconnection at time t.

Let $N_{ij} \overset{df.}{=} \{i', j', h' \mid v^{h'}_{i'j'}$ belongs to the set of output auxiliary vertices of $v_{ij}\}$.

$e_3(i, j, t) \overset{df.}{=} \sum_{N_{ij}} \sum_{u=1}^{d(i',j',t-1)} \delta_{i'j'h'}(Y_u(i', j', t)) =$ production of $V(i, j)$ through disconnection of various $V(i', j')$ (where, if there are two edges of the graph from v_{ij} to $v^{h'}_{i'j'}$, $\delta_{i'j'h'}(Y_u(i', j', t))$ is counted twice in the sum).

$e_4(i, j, t) \overset{df.}{=} -\sum_{N_{ij}} \sum_{u=1}^{C(t)} \delta_{i'j'h'}(X_u(t)) \overset{df.}{=}$ loss of $V(i, j)$ through connection to form various $V(i', j')$ at time t (where, if there are two edges of the graph from v_{ij} to $v^{h'}_{i'j'}$, $\delta_{i'j'h'}(X_u(t))$ is counted twice in the sum).

$d(i, j, t) = d(i, j, t-1) + \sum_{b=1}^{4} e_b(i, j, t)$

The class of generation procedures just defined (assuming an appropriate set of generators has been selected) forms a continuum containing, as a subset, a continuum of universal generation procedures. Given the generation tree and the transition equations of any particular procedure, one can calculate the expected values of the densities as a function of time. From the general form of the transition equations one can determine such things as conditions under which the resulting generation procedures are stationary processes. Derivation of such properties—made possible by the abstract definition—in turn opens the way to results concerning adaptive efficiency. (This subject will be touched upon in section 6). There are many other formulations yielding continua of generation procedures, some simpler and some more intricate; however, few have models as simple as those of the present class. The models will be discussed next.

4. *Models*

Models of the generation procedures of section 3 can be based upon the class of iterative circuit computers (a mathematical characterization of iterative circuit computers has been given in [8]). It should be emphasized at once that the iterative circuit computer is *not* the adaptive system. Rather, the computer is to be identified with the space in which the adaptive system is embedded. The word "space" is used here with a sense close to that of the words "physical space". With a physical space we associate a geometry and a set of universal laws which any particle at any position in that space must obey. Similarly each iterative circuit computer has an associated geometry and a set of "state-transition" rules holding, without change, for each location in the computer. Because iterative circuit computers have been characterized mathematically, we can use this mathematics to deduce theorems about the embedded systems.

Iterative circuit computers, with appropriate interpretation of the symbols, include representatives structurally and behaviorally equivalent to automata of any of the following types: Turing machines (with 1 or more tapes) [8, 15], tesselation automata [23, 12], growing logical nets [2, 1], and potentially-infinite automata [3]. Each computer in the class is constructed of a single basic module (a fixed logical network) iterated to form a regular array of modules. In the mathematical characterization the arrangement of the modules is specified by a finitely-generated abelian group, A, and the scheme of connection of a module to its neighbors is given by a finite set of elements, A^0, selected from A. Any module, being a fixed finite logical net, can assume only a finite number of states. The state of the module at coördinate α at time t will be designated by $S(\alpha, t)$. Each module can control the information that flows through it from its neighbors—each channel to or from a neighbor is in effect gated and at any time, t, any given gate may be either open or closed. Which gates are open and which are closed is specified by a portion, $Y(\alpha, t)$, of the state, $S(\alpha, t)$. Each module also has a fixed storage capacity which may be thought of as a storage register. The other part of the state $S(\alpha, t)$, symbolized by $X(\alpha, t)$, specifies the content of this register. Thus the state at time t of the module at coördinate α is composed of two parts, $S(\alpha, t) = (X(\alpha, t), Y(\alpha, t))$; in the mathematical characterization this means that the set S is the direct product of two finite sets X and Y, $S = X \otimes Y$. Each module also has a fixed number of free inputs which serve as inputs to the computer; the input state of the module at α at time t will be designated by $B(\alpha, t)$. As information flows into a module at time t, through the open channels from its neighbors and through its free inputs, the module processes the information and passes the result along, without delay, through those outgoing channels which are open. The same information is also used to determine what state, $S(\alpha, t + 1)$, the module is to have at the next instant of time. The information used to determine the new storage state, $X(\alpha, t + 1)$, is processed or transformed according to a function, f, called a sub-transition function. The new pattern of open and closed gates at α, $Y(\alpha, t + 1)$, is determined by a function P. Thus the rules for determining the state, $S(\alpha, t + 1)$, of a module are essentially specified by f and P. These rules are

"locally effective". That is, given f and P, and given the states $S(\beta, t)$ and input states $B(\beta, t)$ of selected modules lying within a radius $r(t)$ of α, the state of the module at α at the next instant, $S(\alpha, t + 1)$, is unambiguously determined.

The main properties of iterative circuit computers the reader should keep in mind are:

(1) Once selections are made for A, A^0, S, f, and P, the structure of the basic module and the arrangement of the modules to form the computer are completely determined. Thus the quintuple (A, A^0, S, f, P) completely identifies the iterative circuit computer, and with each distinct quintuple is associated a distinct computer.

(2) In general, an assignment of states to a connected set of modules can be treated as a sub-program stored in the computer. Each of these sub-programs can be active at the same time; thus from the computing point of view, a given iterative circuit computer can execute arbitrarily many sub-programs simultaneously (again within limits imposed by size).

(3) Any given growing automaton (see [1] or [2]) can be simulated by a connected set of sub-programs in the iterative circuit computer.

(4) Important properties which can be given to programs are:

(a) 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.

(b) 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.

(c) Sub-programs can be written so that they can directly produce a copy of themselves (duplicate) in an adjacent set of modules.

(5) 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.

(6) In an iterative circuit computer the primary technique for locating an operand is relative addressing—instead of referring to an operand by an address, a path (a sequence of opened gates) is opened to the operand. Any module belonging to the path can use the storage registers of other modules along the path as operands. As Newell describes the process [14] we point to a location rather than addressing it. Relative addressing permits the set of generators to be quite small.

(7) By suitably restricting the functions f and P it can be arranged that generators do not interpenetrate or "over-write" when moving (cf. the notion of a "billiard ball" physics).

The free generation procedure of section 2, when modelled in an iterative circuit computer, requires the generators (and combinations of generators) to "shift" and "connect" at random in the computer. The simplest form of random shift occurs under the following conditions: (1) At each moment of time a generator has a fixed probability of shifting to one of its neighboring modules. (2) If a generator attempts to shift to a module already occupied by another generator such a shift is prohibited. A similar prohibition is to apply when two generators attempt to shift to the same (initially "empty") module. Under conditions to be discussed further on, two or more generators occupying adjacent modules ("in contact") may become connected. Such connected sets of generators are to shift as a unit and for such sets the above conditions are still to

hold. In particular if a connected set of generators attempts to shift into a set of adjacent modules and any one of these modules is occupied then the shift is to be prohibited.

Conditions (1) and (2) can now be translated into requirements on the iterative circuit computer involved. First of all a portion of the storage register in each module must be used to record the type of generator present (or the absence of any generator). In terms of the characterization this means that the finite set X will be the direct sum of two sets, $X = X_1 \otimes X_r$. If there are k generator types $g_1, \cdots, g_k$ then X_1 will consist of these k elements together with an element g_0 indicating no generator. The complete state of the module at coördinate α and time t is designated by

$$\begin{aligned} S(\alpha, t) &= [X(\alpha, t), Y(\alpha, t)] \\ &= [X_1(\alpha, t), X_r(\alpha, t), Y(\alpha, t)]. \end{aligned}$$

If $X_1(\alpha, t) = g_j$, $1 \leqq j \leqq k$, then we will say that generator g_j is present at coördinate α at time t. The transition equations, f and P, must be chosen so that the two conditions on shifting are satisfied. For example, let generator g_j be present at an immediate neighbor, $\beta = a_i(\alpha)$, of α. Let $X_r(\beta, t)$ indicate that this generator is to shift to position α. Then $X_1(\alpha, t + 1)$ will equal $X_1(\beta, t)$ only if no other immediate neighbor of α indicates a similar shift. Proceeding in this way we can without too much trouble translate all of the earlier requirements into formal requirements on the transition equations. To provide for random shifts we choose a stochastic function, $B'(\alpha, t)$, which for each module α and time t designates a random variable which assumes the value j with probability p_j, $1 \leqq j \leqq n_I$, where n_I is the number of input states. Then we set the input function $B(\alpha, t)$ equal to the jth input state when $B'(\alpha, t) = j$. The question of whether or not a generator shifts to one of its immediate neighbors depends in a completely deterministic fashion on $B(\alpha, t)$ and the inputs from neighboring modules. By giving the transition equations the appropriate form we can assure that the input sequence is used in effect as a source of random numbers for determining the shifts. Thus the probabilistic aspects of the theory are made to depend solely upon the probabilistic properties of the input. Since the usual theorems about deterministic automata are stated in terms of an arbitrary input function, such theorems can still be used directly in the present theory. It should be emphasized again that the procedures just introduced do not preclude systems in which highly sophisticated sub-programs (connected sets of generators) have been set into the computer initially.

The conditions for connection will be stated in terms of two general quantities which will be associated with the generators as components of X: (1) valence and (2) activation. Valence will be specified by part of the X_1 component of X and activation by part of the X_r component. Thus, letting X_{11} specify the instruction type, X_{12} the valence, and X_2 the activation, X will take the form:

$$\begin{aligned} X &= X_1 \otimes X_r \\ &= X_{11} \otimes X_{12} \otimes X_2 \otimes X_{r'}. \end{aligned}$$

Under interpretation the elements of X_{11} are to correspond to instruction types such as ADD, STORE, etc. The object will be to choose f and P to bring about the following desired relations between instruction type, X_{11}, valence, X_{12}, and activation, X_2: Activations are to be treated as integers and accordingly summed and ordered. Let g_i and g_j (elements of X_1) occupy adjacent modules as the result of a shift. Let E be their combined activation. The valence of g_i and g_j determines an interval, $E^*_{ij} = (a, b)$, such that the pair will connect only if E lies in the interval $a \leqq E \leqq b$. To provide for disconnection (essential if there is to be "competition" and differential selection) E^*_{ij} will also be used to determine the probability that g_i and g_j disconnect at any given time after connection. For many models the probability of disconnection will be specified by $(E^*_{ij})^{-1}$; the "stronger" the connection in terms of the activation required for it to form, the lower the probability of disconnection.

The valence-activation combination provides for the introduction of modified generation procedures as follows: Let A be a particular connected configuration of generators and let n be the number of copies of A in some region, V, of the computer. For simplicity, select a particular generator on the periphery of the configuration A and record the activation associated with the corresponding generator in each copy of A. Let $n(E)$ be the number of copies of A having an activation E at the given position. $n(E)$ can be made exponentially decreasing through the following provision: When two generators come into contact (occupy adjacent modules) the corresponding activations E_1 and E_2 are summed, an activation between zero and the sum, $0 < E_1' < E_1 + E_2$, is chosen at random (using $B(\alpha, t)$) to replace the activation E_1, and the activation level $E_2' = E_1 + E_2 - E_1'$ is used to replace activation E_2. (That the resulting distribution of $n(E)$ is exponentially decreasing can be proved as a theorem by applying an argument from statistical mechanics). With an exponential distribution if the activation required for a given type of connection is reduced by half, the number of connections of that type (in region V over time T) will be squared. The problem of modifying the generation procedure thus reduces to one of manipulating the activation required for a given type of connection. However this is not so simple as it might seem. The level of activation required for the connection, E^*_{ij}, is completely specified by the valences of the two generators to be connected. Thus E^*_{ij} cannot be directly manipulated. There is nevertheless an indirect method. This method of control depends upon the introduction of strings of generators—templates—which facilitate connection of other generator strings. (From this point until the end of this section I will assume that we are dealing only with linear strings of generators; the approach can be expanded to arbitrary configurations.) Let us assume that it is desired to increase the rate at which strings of the form A_1g_1 connect to strings of the form g_2A_2 (where g_1 and g_2 are generators, A_1 and A_2 arbitrary strings). Let E^*_{12} be the activation required for g_1 and g_2 to connect (as determined from their valences). Consider now a string of the form $B_1g_1'g_2'B_2$ and let the level of activation required for g_1 to connect to g_1' be $\frac{1}{2}E^*_{12}$, similarly for g_2 and g_2'. This string can be used as a template. Note first that B_1 and B_2 are generator strings and hence, in effect, sub-programs. They

can control alignment when A_1g_1 and g_2A_2 contact the template string; B_1 and B_2 can also select certain subsets of the possible strings A_1 and A_2 as "admissible" so that only when A_1g_1 (or g_2A_2) involves an admissible A_1 (or A_2) is the process allowed to continue. If the pair (g_1 , g_1') has a total activation $E = \frac{1}{2}E^*_{12}$, then A_1g_1 will connect to the template. A similar statement holds for the pair (g_2 , g_2') and the string g_2A_2. Once the two strings are connected to the template, B_1 and B_2 can use the activation involved, $\frac{1}{2}E^*_{12} + \frac{1}{2}E^*_{12} = E^*_{12}$, to form a direct connection between A_1g_1 and g_2A_2 in place of the connections to the template. That is, B_1 and B_2 will "erase" the connections between g_1 and g_1', g_2 and g_2', and "transfer" the total activation involved, E^*_{12}, to g_1 and g_2. Thus, enough activation is provided for a connection between g_1 and g_2. The net result is that $A_1g_1g_2A_2$ is formed and the template is freed to repeat the process (cf. the action of a catalyst or enzyme). Because only $\frac{1}{2}E^*_{12}$ is required, the number of connections of A_1g_1 (similarly g_2A_2) to the template will be the square of the number of connections it would make on direct contact with g_2A_2. The template thus has the desired effect of increasing the rate of connection of the two strings. This of course is only a sketch of one way in which such "templates" can alter the rate of connection—the intention of the sketch is to indicate that such control is possible and that it fits within the framework so far presented.

Templates exert an effect far out of proportion to their numbers because: (1) the exponential distribution of activation amplifies the effect of individual templates, (2) templates modify connection processes without being altered themselves (the catalyst effect), (3) in modifying the generation rate of a given type of program, the template also affects the generation rate of all programs which use the given program as a component. Thus, by introducing programs which construct templates—supervisory programs—one gains extremely flexible control over the generation procedure. In effect the supervisory program modulates the free generation process through the amplification factor of the templates it produces.

The procedure described in section 2 requires that the supervisory program duplicate, with some probability of variation or mutation, upon accumulation of sufficient activation. Briefly, duplication amounts to the supervisory program copying itself into an adjacent set of modules. The likelihood of variation at a given position in the supervisory program can be made a function of the E^*_{ij} and the random inputs $B(\alpha, t)$ (other procedures exist). Operation now proceeds as described in section 2 if the supervisory program, in order to duplicate, is required to collect from the environment an amount of activation equal to the activation involved in its own connections. The nature of the environment and the way in which the supervisory-template-solver complex adapts to it will be discussed in the next section.

5. *Environment and Adaptation*

In section 2 it was suggested that the environment could be treated as a population of problems. For present purposes let us restrict the problems to well-

defined problems—problems which are presented by means of a finite set of initial statements (statement of the problem) and an algorithm for checking whether a purported solution of the problem is in fact a solution. For example, the initial statements could specify an axiom system and a theorem to be proved therein; a tentative solution would be any sequence of statements which purports to be a proof of the theorem; the checking algorithm would be the usual routine which checks that each statement in the sequence follows from previous statements by the allowed rules of inference, with the last statement being the one to be proved. Well-defined problems can easily be embedded in the space defined by the iterative circuit computer. The checking algorithm becomes a program and the initial statements are coded as configurations of generators. The problem is solved when a problem-solving program transforms the coded initial statements into a configuration which the checking program accepts as a solution. In accordance with other suggestions in section 2, each embedded problem will be assigned a quota of activation (the same quantity that was involved in the connection processes discussed in section 4). This activation is released to the controlling supervisory program whenever a problem-solver solves the embedded problem.

When we consider the interaction of an adaptive system with its environment we come very soon to questions of partial solutions, sub-goals, etc. Here such questions become questions about the relation of supervisory programs to the checking routines of embedded problems. One of the simplest cases occurs when there is an *a priori* estimate of the nature of a partial solution and, perhaps, a measure of the closeness of its approach to the final solution. (Such *a priori* estimates are quite common in the problem-solving programs constructed to date; see Samuel [16], Gelernter and Rochester [6], and particularly Newell, Shaw, and Simon [13].) To make use of these estimates we simply incorporate them in the checking routines; the checking routine is written so that some of the activation associated with the problem is released whenever a partial solution is presented.

A more interesting question of partial solution occurs when the environment involves stochastic sequences, the problem being to predict values of the sequence at a future time. (The stochastic sequence may be produced by an embedded sequence generator or it may be a component of the input sequence $B(\alpha, t)$; the checking program simply matches the prediction for time $t + j$ against the actual value at $t + j$ and releases activation as a function of the "closeness" of the prediction). Note that even a program which uses random number generation as its mode of "prediction" will succeed in releasing some activation. However, programs that improve upon this procedure, by utilizing some of the data to produce better than chance predictions, will gain a selective advantage for the corresponding supervisory programs. The adaptive system can accumulate subroutines which improve its predictions; these can in turn be combined in various ways to generate still more sophisticated prediction procedures. Thus, the adaptive system is not required to leap to "the" solution (if any), but can approximate it by stages.

The prediction problem is one example of a "rich" environment. In the simplest sense an environment is rich when it is composed of graded sequences of problems. By its very nature, such an environment permits the adaptive system to develop subroutines for problems at one level of difficulty which, through recombination, minor changes, insertion, etc., will be useful in solving problems at the next level of difficulty. Operations such as recombination involve only minor modification of the generation procedure; thus, the "tuning in" procedure mentioned at the end of section 2 becomes very effective. Faced with a rich environment, the generation procedure can be adjusted in a continuous fashion to ever more difficult segments of the environment. In effect, the adaptive system can accumulate a repertoire of programs which serve as a new set of generators for programs aimed at problems of the next level of difficulty. Under these conditions the supervisory programs can develop problem-solving programs which are both complex and general in a surprisingly short time. For a rough indication, compare the number of j-step-programs on k generators, k^j, with the number when each level of the hierarchy requires just h-step-programs on k generators, $k^h(\log_k j)$.

Rich environments are not rare. Frequently problem environments which closely approximate some natural situation will prove to be rich in the above sense. The problem of predicting future states of a natural system (e.g. a weather prediction problem) can often be given the format of the prediction problem just discussed. One further example, the general question of stability (survival) of biochemical systems, may give some hint of the profusion and variety of rich environments. In the present context this becomes a question concerning embedded automata. The generators, connections, activation, templates, etc., become the abstract (and highly idealized) representatives of atoms, bonds, activation, enzymes, etc., in the biochemical system. Stability becomes preservation of structure in the face of spontaneous disconnection. Stability is a matter of degree; for an initial random mixture of unconnected generators some early stages of increasing stability are: (i) aggregates of generators formed by randomly occurring connections and disconnections; (ii) restricted aggregates with only selected generators being admitted to the system; (iii) self-modifying, restricted aggregates wherein structural changes in one part of the aggregate are induced by another part; (iv) catalysed aggregates where the rate of aggregation and modification is controlled by templates. Eventually, this sequence should lead to: (n) aggregates with supervisory control and self-reproduction. The more stable system, by its very nature, will have a higher expected density (in space-time) than its less stable companions. Therefore, trial formation of any new. potentially more stable system will incorporate previously formed systems with a probability related to their stability.

Mathematical characterization of classes of "rich" environments relative to a given class of adaptive systems constitutes one of the major questions in the study of adaptive systems.

The advantages pertaining to a rich environment suggest that an adaptive system could enhance its rate of adaptation by somehow enriching the environ-

ment. Such enrichment occurs if the adaptive system can generate subproblems or subgoals whose solution will contribute to the solution of the given problems of the environment (cf. Newell, Shaw and Simon [13] or Minsky [11]). The simplest way to bring this idea into the present context is to allow supervisory programs to cause "auxiliary" checking routines to be formed. Since a checking routine is after all just a program, it is certainly possible for a supervisory program to bring about its formation. Such auxiliary checking routines constitute well-defined problems introduced by the supervisory program. This amounts to the introduction of a "law of effect": Each time a subgoal is achieved and then a given problem subsequently solved, the procedures for forming the relevant subgoal are rewarded by survival. In greater detail: A given supervisory program, under the stated conditions, will implicitly define both a population of problem-solving programs *and* a population of auxiliary checking routines. This mixed population can be expected to release activation from the given well-defined problems at some rate, $r(t)$. Some of this released activation, say $r'(t)$, will be tapped off to supply activation to the auxiliary checking routines. The remainder, $r(t) - r'(t)$, will be channeled to the supervisory program to be used eventually in its duplication. The net rate of accumulation of activation at the supervisory program, $r(t) - r'(t)$, determines its relative advantage under differential selection. For example: Let $r_0(t)$ be the rate of release of activation obtained by the problem-solving population of a supervisory program S_0 which generates no subgoal problems. Let $e_1(t)$ be the change in the rate of release obtained by a supervisory program S_1 which differs from S_0 only in that it generates some subgoal problems too. Let $r_1'(t)$ be the activation tapped off for the subgoal problems of S_1. Then S_1 will have a selective advantage over S_0 only if

$$[r_0(t) + e_1(t) - r_1'(t)]_{\text{ave}} > [r_0(t)]_{\text{ave}}.$$

In more intuitive language, subgoals confer a selective advantage only if they do not cost too much relative to the additional problem-solving capability they provide.

6. *Comment*

In the framework outlined, the problem of adaptation becomes one of modifying a free generation procedure in order to maximize activation release from the environment. However, the modification can only be carried out in terms of available information about the environment. Moreover, this information (unless it is given a priori) can only be obtained from trials or experiments performed on the environment. Thus the rate of adaptation is limited by the rate of information accrual. In fact, for certain classes of generation procedures, the maximum justifiable modification (skewing) of a free generation procedure has been determined as an explicit function of the accumulated information. (The proof of this and some related theorems will be published in another paper.) For such systems, it can also be shown that two apparently distinct criteria for determining the amount of skewing are identical: The first criterion selects

that amount of skewing which, on the basis of available information, maximizes the expected activation release; the second criterion adjusts the skewing, on the basis of available information, according to a minimax estimate of the environment. In other words, for the systems described, the most opportunistic modification of the generation procedure is at one and the same time the most conservative—a somewhat surprising corollary.

The theorem concerning maximum justifiable modification is in effect an "efficiency" theorem for adaptive systems . . . it sets a limit on adaptation in terms of information. In analogy with other "limiting" theorems (such as the channel capacity theorems) one would like to establish a companion "relizability" theorem. That is, one would like to exhibit a sequence of models with efficiencies which approach the efficient limit asymptotically. The iterative circuit computer models (section 4) were developed with this in mind, but there still remains a great deal to be done in the investigation of asymptotically efficient sequences of models.

The central purpose of this paper has been to suggest how questions concerning adaptation can be treated in a logical context. There are, however, the larger questions of whether the theorems within reach will be useful and whether one should attempt a theory at all (rather than contribute, say, to the growing body of heuristics concerned with adaptation). Von Neumann has summed up the situation quite clearly:

> "(The theory's) first applications are necessarily to elementary problems where the result has never been in doubt and no theory is actually required. At this early stage the application serves to corroborate the theory. The next stage develops when the theory is applied to somewhat more complicated situations in which it may already lead to a certain extent beyond the obvious and the familiar. Here theory and application corroborate each other mutually. Beyond this lies the field of real success: genuine prediction by theory." [21]

Acknowledgments. I would like to thank Professor Arthur Burks and Dr. Jesse Wright for listening to a reading of the first draft of this paper and for making many helpful suggestions concerning the exposition. I would also like to thank Dr. David Willis and Dr. Richard Tanaka for the opportunity to begin the writing of this paper while at Lockheed Missile and Space Division on leave from the University of Michigan. Finally, I would like to thank students in the Program in Communication Sciences at the University of Michigan for their questions and discussion in a course where many of the ideas in this paper were first presented. This work has been supported at various stages by the National Science Foundation through grants G-4790 and G-11046 and currently by the U. S. Army Signal Corps through contract DA-36-039-sc-87174.

REFERENCES

1. Burks, A. W Computation, behavior and structure in fixed and growing automata In *Self-Organizing Systems*, Pergamon Press (1960).
2. Burks, A. W., and Hao Wang. "The Logic of Automata," *J. ACM 4*, (1957) 193–218, 279–297.
3. Church, A. Application of recursive arithmetic to the problem of circuit synthesis.

In Summer Institute for Symbolic Logic Summaries, Institute for Defense Analysis (1957).

4 CRAIG, W. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *J. Symbolic Logic 22*, No. 3, (1957) 250–268.

5. FISHER, R. A. *The Genetical Theory of Natural Selection*. Dover (1958).

6. GELERNTER, H. L., and ROCHESTER, N. Intelligent behavior in problem-solving machines. *IBM J. Res. Dev. 2*, No. 4, (1958) 336–345.

7. HOLLAND, J. H. A universal computer capable of executing an arbitrary number of sub-programs simultaneously. Proc. Eastern Joint Comp. Conf. (1959) 108–113.

8. HOLLAND, J. H. Iterative circuit computers. Proc. Western Joint Comp. Conf. (1960) 259–265.

9. JAKOWATZ, C. V.; SHUEY, R. L.; and WHITE, G. M. Adaptive waveform recognition. GE Research Lab Report 60-RL-2435 E (1960).

10 LYNDON, R. C. An interpolation theorem in the predicate calculus. *Pacific J. Math. 9*, No. 1, (1959) 129–142.

11. MINSKY, M. "Steps Toward Artificial Intelligence," *Proc. IRE, 49*, No. 1, (1961) 8–30.

12. MOORE, E. F. Machine models of self-reproduction. Paper presented at meeting of Amer. Math. Soc , Cambridge, Mass. October, 1959.

13. NEWELL, A.; SHAW, J. C.; and SIMON, H. A. A variety of intelligent learning in a general problem solver. In *Self-Organizing Systems*, Pergamon Press (1960).

14. NEWELL, A. On programming a highly parallel machine to be an intelligent technician. Proc. Western Joint Comp. Conf. (1960) 267–282.

15. RABIN, M. O., and SCOTT, D. Finite automata and their decision problems. *IBM J. Res. Dev. 3*, No. 2, (1959) 114–125.

16. SAMUEL, A. L. Some studies in machine learning, using the game of checkers. *IBM J. Res. Dev. 3*, No. 3, (1959) 210–229.

17. SELFRIDGE, O. G. Pandemonium, a paradigm for learning. In *Mechanization of Thought Processes*, Nat'l. Phys. Lab. Symp. No. 10, Her Majesty's Stationery Office (1959).

18. TURING, A. M. On computable numbers, with an application to the entscheidungsproblem. *Proc. Lond. Math. Soc.* (2), *43*, (1936) 230–265.

19. TURING, A. M. Computing machinery and intelligence. *Mind 59*, (1950) 433–460.

20. TURING, A. M. The chemical basis of morphogenesis, *Phil. Trans. Roy. Soc.* ser. B *237*, (1952) 37 ff.

21. VON NEUMANN, J. and MORGENSTERN, O. *Theory of Games and Economic Behavior*. Princeton (1947).

22. VON NEUMANN, J. The general and logical theory of automata. In *Cerebral Mechanisms in Behavior—The Hixon Symposium*. Wiley (1951).

23. VON NEUMANN, J. The theory of automata. Construction, reproduction, homogeneity. Unpublished manuscript.

24. WRIGHT, S. Physiological genetics, ecology of populations, and natural selection In *The Evolution of Life*, Chicago (1960).