Scan A258

Hogg + Huberman

add to many

R

PRV A32 2342 85.

fa1

258
307
357
405
1669
1765
5945
~~5946~~
5949

Attractors on finite sets: The dissipative dynamics of computing structures

T. Hogg and B. A. Huberman
Xerox Palo Alto Research Center, Palo Alto, California 94304

(Received 23 January 1985)

We present a theory of attractors on finite sets which is applicable to finite-state systems such as computing structures and other systems which display a hierarchy of organizations with a discrete time evolution. Because computing with attractive fixed points can lead to reliable behavior [B. A. Huberman and T. Hogg, Phys. Rev. Lett. 52, 1048 (1984)], the theory deals with dissipative processes, i.e., those which contract volumes in phase space. The stability of such systems is quantified and analytic expressions are obtained for the appropriate indices in some limiting cases. It is also shown that trees with ultrametric topologies provide the natural language for these systems. The theory is extended to include several practical constraints, and connections are made with experimental quantities which can be measured in particular architectures.

H^n_m: mappings from n elements to 1 in m steps

I. INTRODUCTION

The time evolution of computing structures poses interesting challenges when trying to develop a science of their behavior. A main difficulty appears when one attempts to use conventional dynamical systems theory. Being finite-state systems that operate in digital space-time, computers cannot be described with the usual concepts based on continuum dynamics and classical analysis. This inherent problem is illustrated by the familiar notion of generic stability, which is based on the relative behavior of trajectories which differ initially by arbitrarily small amounts.[1] With finite-state machines, such a limit does not exist because of the intrinsic granularity of their phase spaces. Rather, one is forced to consider dynamics on a totally different topology which is appropriate for the time evolution of finite sets. Therefore, a proper language with which to codify such behavior is given by combinatorial analysis and discrete maps.

This paper presents a theory of attractors on finite sets which is applicable to finite-state systems such as computing structures and other systems which display a hierarchy of organizations with a discrete time evolution. Because reliable behavior is intrinsically interesting, and can appear when dealing with attractive fixed points,[2] we will concentrate on dissipative processes, namely those which contract volumes in phase space. Furthermore, we will restrict both the space-time and state variables to a finite set of values, which is the case in the real world. This is to be contrasted with the general theory of cellular automata and formal Turing machines, where most of their interesting properties depend on their being infinite. This restriction leads in turn to the introduction of trees with ultrametric topologies, which are the appropriate language for these systems.[3] Such a language also provides a natural framework for the discussion of the emergence of global computational capabilities out of a collection of simpler units. We should mention that with minor modifications the whole theory will apply to the deterministic dynamics of Boolean networks.[4]

Section II presents the theory of dissipative discrete dynamics on finite sets, as well as a diagrammatic technique which is useful for analyzing the behavior of maps on finite sets. In Sec. III the theory is extended to include several practical constraints and Sec. IV presents some illustrations of the results obtained in the previous sections. A concluding section summarizes the main points of the theory, and outlines its extensions to adaptive structures. An appendix presents the derivation of the generating function for the H numbers, which measure the number of discrete trajectories which map given inputs into attractive fixed points in a fixed number of iterations.

II. GENERAL THEORY

In this section we will describe the basic properties of deterministic discrete dynamical systems and consider a number of simple examples to illustrate these ideas.

(a) Maps on finite sets. The dynamics of a quantity that takes on a finite set of values and changes only at discrete instants of time is governed by discrete maps. To investigate the nature of such processes we will consider maps of a finite set S into itself. Two simple examples of such mappings, for the set of integers $S=\{0,1,2,3\}$, are given in Fig. 1. In the map of Fig. 1(a), four inputs are

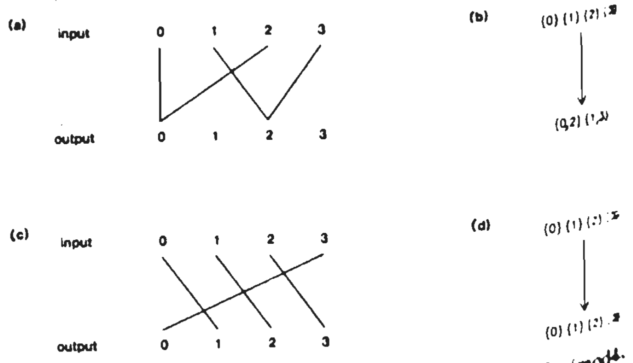

FIG. 1. (a) The map on $\{0,1,2,3\}$ given by $f(n)=2n$ (mod4). (b) Equivalence classes for this map. (c) A permutation on $\{0,1,2,3\}$ given by $0\to1\to2\to3\to0$. (d) Equivalence classes for the permutation.

mapped to two outputs, thereby giving a contraction factor of 2. The map can alternatively be characterized by the sets of inputs that are mapped to each output which in this case are $\{0,2\}$ and $\{1,3\}$, as shown in Fig. 1(b). These sets define equivalence classes such that the result of the map acting on each member of a class is the same. The other example, shown in Fig. 1(c), is a permutation and therefore has no contraction in its dynamics. In this case each input is mapped to a unique output and hence is in a separate equivalence class as shown in Fig. 1(d). In these examples, the original inputs can be considered as a collection of singleton classes, in that at the input stage each input is distinct.

These equivalence classes form a partition of the original set of inputs S, i.e., the classes do not have any elements in common and their union is the whole set S. Alternatively, a map can be represented by a collection of trees of depth 1 in which all points that are mapped together are considered nodes in the same tree. This tree structure provides a simple graphical representation of the map, examples of which are given in Figs. 1(a) and 1(c). A contractive map, i.e., one that takes many inputs into fewer outputs, is the discrete analog of a dissipative physical system which contracts volumes in phase space. The asymptotic behavior of these systems is determined by their attractors and associated basins of attraction. For contractive maps, the inputs to each tree, or the members of each equivalence class, specify the basins of attraction for the system. We can then define a global contraction rate C in terms of the number of inputs (N_{input}) and the final number of outputs (N_{output}) as

$$C = N_{\text{input}}/N_{\text{output}} . \tag{1}$$

In the remainder of this paper we will focus on the nature of discrete attractors regardless of the actual value of any given output. Thus maps that produce the same equivalence-class structure will be considered equivalent even though the actual outputs may differ in their values.

To generate dynamics over time periods longer than the single iteration we have just discussed, these maps can be iterated many times, possibly with a different map at each step. The result after m steps can be viewed as a collection of trees of uniform depth m, with the leaves representing the original inputs and the roots representing the final outputs. The branching at each node gives the number of distinct inputs that are mapped to the same output at that step of the series of maps. Examples of such structures are shown in Fig. 2(a). Or, viewed in terms of equivalence classes, the maps specify successive partitionings of the original set of inputs in which the classes resulting from the map at step k are unions of classes at step $k-1$. That is, the sets of inputs that can be considered equivalent after k steps (because they produce the same output at this point) are formed by combining the corresponding sets at the $(k-1)$th step. Furthermore, by disregarding the particular outputs and focusing instead on these sets, each series of maps is uniquely identified by the corresponding series of successive partitionings. We thus have three equivalent ways of viewing discrete dynamical systems: (1) as iterated maps, (2) as trees, or (3) as successive collections of equivalence classes. Contractive maps will then correspond to trees with many leaves, or to equivalence classes that become larger as the system evolves in time.

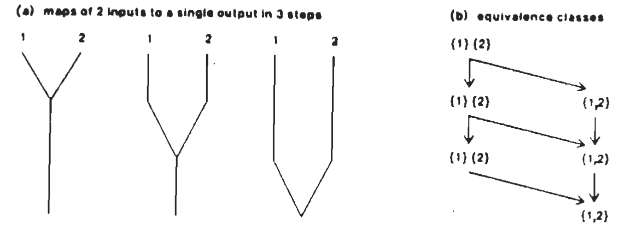

FIG. 2. (a) The three ways of mapping two inputs to a single output in three steps represented as trees. (b) Corresponding equivalence classes. Each of the three paths through the classes corresponds to a specific map.

Before proceeding further, we should point out that these trees possess an ultrametric topology.[5] By this we mean that a distance d can be defined such that any three nodes on the same level form an isosceles triangle. Specifically, these nodes can be labeled in such a way that $d(a,b)=d(b,c)\geq d(a,c)$. For the trees corresponding to iterated maps, an appropriate ultrametric distance between nodes at given level of a single tree is the number of steps the maps are iterated from that point on before they produce the same output. Unlike the usual metrics such as Hamming distance[6] which depend only on the inputs, this one is generated by the maps themselves. It thus becomes the natural distance with which to analyze contractive dynamics of finite sets.

To clarify the behavior of these maps, we will first examine the characteristics of a single basin of attraction, i.e., maps that produce a single output, and for definiteness, consider sets of integers. Specifically, given a dynamical system with a specified global rate of contraction (or dissipation) C, we will now compute $H^n{}_m$, the total number of ways the set $S=\{1,2,\ldots,n\}$ can be mapped into a single output in m time steps. This number, and the methods used to compute it, will then be used to quantify the behavior of these maps. Alternatively, $H^n{}_m$ is the total number of paths that take the original (singleton) equivalence classes $\{1\},\{2\},\ldots,\{n\}$ into a single final class $\{1,2,\ldots,n\}$ in m steps. Note that the global contraction rate C is given by n. For the sake of simplicity we will first consider the case of two inputs, as it will illustrate some of the diagrammatic techniques which we will use throughout this paper.

H^n_m

Consider two inputs being mapped through m steps into a given output. The possible set of trajectories that take inputs from $m-1$ to m are shown in Fig. 2(a) for the case of $m=3$. In the language of equivalence classes, in order to group the two singletons $\{1\}$ and $\{2\}$ into a doubleton $\{1,2\}$ in m steps, one can either group them by the time the $(m-1)$th step is reached in $H^2{}_{m-1}$ ways, or else wait until the last step to join them. This is shown in Fig. 2(b). Therefore, the total number of paths at row m is given in terms of the number of paths at row $m-1$ by the following recursion relation

$$H^2{}_m = H^2{}_{m-1} + 1 \tag{2}$$

with $H^2{}_1 = 1$. That is, each new step introduces one additional possible map. This simple recursion relation has the solution

$$H^2{}_m = m \tag{3}$$

which shows that the number of paths for two inputs increases linearly with m, the number of steps.

In an analogous fashion one can calculate the number of paths that take three inputs into a fixed point in m steps. Figure 3 illustrates the possible dynamical configurations in terms of equivalence classes and is an extension of Fig. 2(b) to the case of three inputs and m steps. That is, the final equivalence class $\{1,2,3\}$, can be formed from five different partitions, namely, $\{1,2,3\}$, $\{1\}\{2,3\}$, $\{2\}\{1,3\}$, $\{3\}\{1,2\}$, and $\{1\}\{2\}\{3\}$. $H^3{}_m$ will then be the sum of the number of ways these partitions can each be formed in $m-1$ steps. For instance, the partition $\{1\}\{2,3\}$ can be formed in $H^1{}_{m-1}H^2{}_{m-1}$ ways because it contains a singleton set and a set with two elements. Thus the required recursion relation is

$$H^3{}_m = H^3{}_{m-1} + 3H^1{}_{m-1}H^2{}_{m-1} + (H^1{}_{m-1})^3 \; . \tag{4}$$

Because $H^2{}_m$ is given by Eq. (3), this can be solved to give

$$H^3{}_m = m(3m-1)/2 \tag{5}$$

showing that the number of paths for three inputs increases quadratically with the number of steps. For instance, there are five ways of mapping three inputs to a single output in two steps, and the corresponding trees are shown in Fig. 4.

(b) Characterizing the attractors. A series of iterated maps, or its resulting tree structure, can be characterized by its response to perturbations or errors in the input. As in the case of continuous physical systems, this provides a measurement of the stability or reliability of the system. However, discrete dynamics differs from its continuous counterpart in that arbitrarily small perturbations are not possible. Instead, perturbations, if they recover, will do so *exactly* and within a finite number of time steps. This behavior in turn requires a modification of the usual quantities that describe global dynamical behavior. In the remainder of this section we will evaluate the probability for a perturbation to heal, in the sense that the perturbed and original inputs produce the same final output. We will also compute the tumbling index Λ for those errors

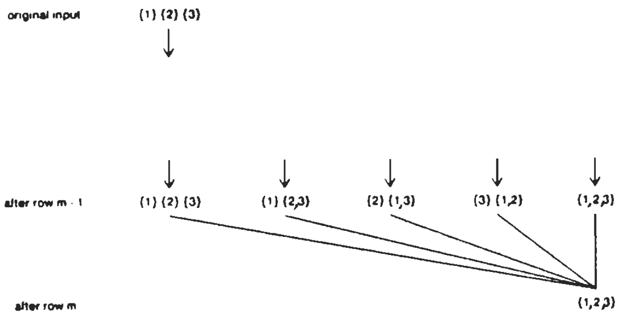

FIG. 3. Possible equivalence class groupings for three inputs mapped to a single output in m steps.

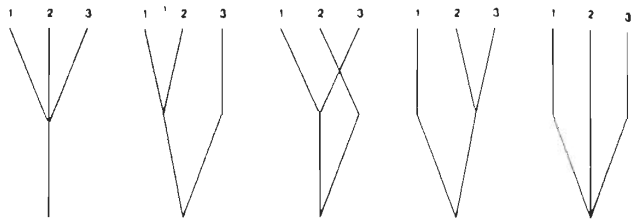

FIG. 4. The five maps of three inputs into a single output in two steps represented as trees.

that do eventually heal, which gives the number of steps required for the recovery to take place.

Specifically, for a set of trees T produced by a particular series of maps and an input i, let $P(T,i,i')$ be the probability that another input i' gives the same output as i. This is 1 if i' is in the same tree as i, and 0 otherwise. The stability of i with respect to errors can be characterized by averaging over all possible i' to give

$$P(T,i) = (1/n)\sum_{i'} P(T,i,i') = |t_i|/n \; , \tag{6}$$

where $|t_i|$ is the number of inputs in the same tree as i (i.e., the number of leaves of the tree) and n is the total number of inputs. Alternatively, $|t_i|$ is the size of the basin of attraction containing i. Finally, the behavior of the series of maps itself (or the corresponding collection of trees T) is characterized by averaging over all of the inputs, i.e.,

$$P(T) = (1/n)\sum_i P(T,i) = (1/n^2)\sum_{t\in T} |t|^2 \; , \tag{7}$$

where the latter sum is over all trees t in the collection T and $|t|$ is the number of inputs, or leaves, of t.

Similarly, the behavior of those errors that do eventually recover is characterized by the tumbling index Λ, which measures the number of steps before the original and erroneous inputs map to the same point.[2] One can also view Λ as giving the recovery time, or ultrametric distance, between any two inputs in the same tree. Specifically, if i and i' are in the same tree then $\Lambda(T,i,i')$ is the ultrametric distance between i and i'. For example, in the three trees of Fig. 2(a), the distances between inputs 1 and 2 are 1, 2, and 3, respectively, which is just the number of steps through the corresponding map required before the two inputs produce the same output. Again the overall behavior can be characterized by averaging:

$$\Lambda(T,i) = (1/|t_i|)\sum_{i'\in t_i} \Lambda(T,i,i') \; , \tag{8}$$

where the sum is over all inputs in the same tree as i, and

$$\Lambda(T) = (1/n)\sum_i \Lambda(T,i) \tag{9}$$

summed over all inputs i. Note that Λ can range between 0 and m, the number of mapping steps (or the number of levels in the tree other than the root). A small value of Λ means that errors in the input heal very quickly, if they heal at all, whereas a large value indicates a long healing length.

We should also mention that this discussion can be applied to the case in which errors occur within the tree instead of at the input stage. Thus if the perturbation is applied after k steps in an m-step sequence of maps, its behavior will be determined as above except that there will be only $m-k$ remaining steps instead of the original m. In this way one can quantify both a healing length and a healing time for perturbations that occur at any point in the dynamical system. An indication of the long-time stability of the maps is given by comparing this healing time to the rate at which errors are introduced. When the healing time is very short, the system will generally be able to recover between errors and thereby maintain its state. In the other limit, errors could build up and eventually cause the outputs to be in another basin of attraction.

Although we have focused on the error-recovery properties of the attractors, it is important to note that there will be other factors of interest in real systems. To see this we can consider two extreme cases, namely the identity map and the map which takes all inputs to a single output. In the first case, Λ is zero since each input is in a separate tree but P will be very small since any perturbation will give a different output. Thus the healing length is very small but the system is extremely sensitive to errors. In the second case, P will be 1 since all inputs produce the same output and the tumbling index will be close to 1. These examples span the range of our reliability measures, but neither exhibits interesting or complex global behavior. Thus the maps involved in most cases of interest will have intermediate values of P and Λ.

(c) Examples. In simple cases these quantities can be explicitly computed. One such example arises when there are τ balanced binary trees with m rows so that a total of $n=2^m\tau$ inputs are mapped to τ outputs. This is illustrated in Fig. 5 for the case in which $\tau=2$ and $m=2$. Since all the trees are the same size we have $P(T,i)=P(T)=1/\tau$. The uniform structure of the trees also implies that $\Lambda(T,i)$ is independent of i, so that $\Lambda(T,i)=\Lambda(T)$. To evaluate Λ, consider input number 1 in Fig. 5. There are four inputs in the corresponding tree so that from Eq. (8)

$$\Lambda(T,1)=\tfrac{1}{4}[\Lambda(T,1,1)+\Lambda(T,1,2)+\Lambda(T,1,3)+\Lambda(T,1,4)]$$
$$=\tfrac{1}{4}(0+1+2+2)=\tfrac{5}{4}\ . \qquad (10)$$

Thus $\Lambda=\frac{5}{4}$ for this set of trees. Similarly, for arbitrary m and a particular input i, there will be one input at distance 0 (namely, i itself), one at distance 1, two at distance 2, and so on up to 2^{m-1} at distance m. Thus

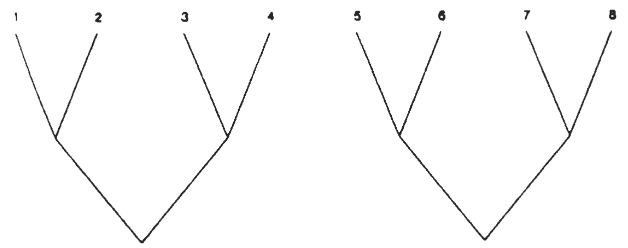

FIG. 5. Example of two balanced binary trees corresponding to a two-step map. Each tree has four inputs and the total number of inputs is $n=2\times2^2=8$.

$$\Lambda(T)=2^{-m}(0+1+2\times2+3\times4+\cdots+m\times2^{m-1})$$
$$=m-1+2^{-m}\ . \qquad (11)$$

For $m=2$, this gives $\frac{5}{4}$ as obtained previously in Eq. (10). Note that for these maps, Λ is nearly equal to m, its largest possible value, so the healing of errors within a balanced binary tree is very slow, i.e., most errors are not corrected until the last few steps.

As another example, consider the case in which the branches at each step come from a single node, with all the branching ratios the same. Specifically, suppose that there are τ such trees with branching ratio b so that the total number of inputs is $n=\tau b$. Figure 6 shows the situation for $\tau=3$ and $b=5$. As in the previous case, the trees all have the same size and structure. Thus $P(T,i)=P(T)=1/\tau$ and

$$\Lambda(T,i)=\Lambda(T)=(1/b)[0\times1+1\times(b-1)]=1-1/b\ . \qquad (12)$$

In this example, Λ is small, indicating that errors within the inputs of a single tree recover rapidly. Note also that Fig. 6 illustrates the full tree structure at each step and not just at the input. The large contraction at each step will also give rapid recovery from errors that occur at any step in the dynamics.

(d) The general case. The calculation of the number of maps given above for cases involving a small number of inputs can be generalized to an arbitrary number of inputs and also to the computation of the tumbling index Λ. In the general case of n inputs, the recursion relation can be derived by considering the situation shown in Fig. 7. That is, a typical partition of the full set will have λ_1 singleton sets, λ_2 sets with two elements, etc., and λ_n sets with n elements. Note that the number of elements in all of the sets must add to n, i.e.,

$$\sum_{i=1}^{n} i\lambda_i=n\ . \qquad (13)$$

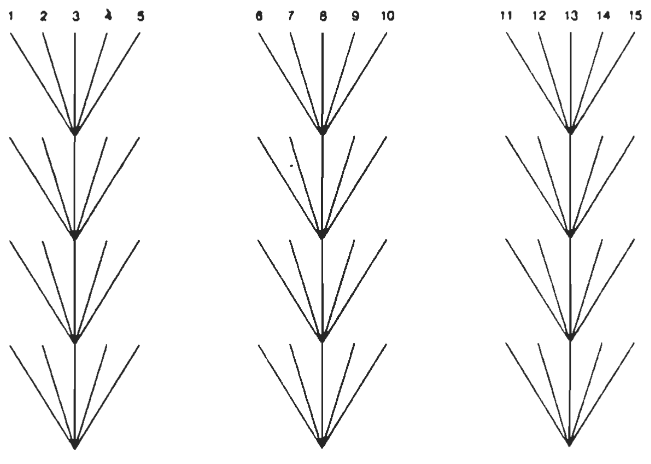

FIG. 6. Example of three balanced trees corresponding to a four-step map. Each tree has five inputs and the total number of inputs is $n=15$.

$$\begin{tikzcd}
\overbrace{\{x\}\{x\}\ldots\{x\}}^{\lambda_1}\ \overbrace{\{xx\}\{xx\}\ldots\{xx\}}^{\lambda_2}\ \overbrace{\{xxx\}\ldots\{xxx\}}^{\lambda_3}\ \ldots\ \overbrace{\{xx\_x\}}^{\lambda_n} \arrow[d] \\
\{1,\_,n\}
\end{tikzcd}$$

FIG. 7. A typical partition of the set $\{1,2,\ldots,n\}$ containing λ_1 singleton sets, λ_2 sets with two elements, etc., and λ_n sets with n elements. Each x is to be replaced with a unique value from 1 to n. Note that the λ_i's must satisfy $\sum_{i=1}^{n} i\lambda_i = n$.

Thus a typical m-step map will correspond to a partition of this form after the $(m-1)$th step and the last step will create the final equivalence class $\{1,2,\ldots,n\}$. Each set in this partition of size k can be formed from the original inputs in $m-1$ steps in $H^k{}_{m-1}$ ways. Thus this particular partition can be produced by a total of

$$(H^1{}_{m-1})^{\lambda_1}(H^2{}_{m-1})^{\lambda_2}\times\cdots\times(H^n{}_{m-1})^{\lambda_n} \tag{14}$$

different maps. The number of such partitions is given by a modified multinomial coefficient $[n;\lambda_1,\lambda_2,\ldots,\lambda_n]$ which equals[7]

$$[n;\lambda_1,\lambda_2,\ldots,\lambda_n] = \frac{n!}{\lambda_1!\lambda_2!\times\cdots\times\lambda_n!(1!)^{\lambda_1}(2!)^{\lambda_2}\times\cdots\times(n!)^{\lambda_n}} . \tag{15}$$

This relation can be seen from the typical partition shown in Fig. 7. There are $n!$ ways of dividing the inputs among the various sets but this overcounts since the ordering of the sets of equal size, as well as the elements within each set, does not matter. For example $\{1,2\}\{3,4\}$ is the same partition as $\{3,4\}\{1,2\}$ and $\{1,2,3\}$ is the same as $\{3,1,2\}$. Since there are λ_k sets of size k, one must divide by $\lambda_k!$ to avoid counting different orderings of the sets as distinct. Similarly, there are $k!$ ways to order the elements within a set of size k so that a factor of $k!$ must be divided out for each such set. This gives a divisor of $(k!)^{\lambda_k}$ for all sets of this size. Combining these factors then gives the actual number of distinct partitions in Eq. (15). The final recursion relation for $H^n{}_m$ can be obtained from Eqs. (14) and (15) by summing over all possible partitions of $\{1,2,\ldots,n\}$. Thus

$$H^n{}_m = \sum [n;\lambda_1,\lambda_2,\ldots,\lambda_n] \prod_{k=1}^{n} (H^k{}_{m-1})^{\lambda_k} , \tag{16}$$

where the sum is over all λ_i's satisfying Eq. (13) and $H^n{}_1 = 1$. Although this appears to be a complex nonlinear recursion relation, the highest term on the right-hand side, $H^n{}_{m-1}$, only enters linearly from the term corresponding to $\lambda_n = 1$ (and the other λ_i's equal to zero). Thus $H^n{}_m$ can be obtained from the values of $H^k{}_m$ for $k<n$ by solving an inhomogeneous *linear* recursion relation. In this way it can be shown that $H^n{}_m$ is a polynomial in m of degree $n-1$ whose leading term is $n!(m/2)^{n-1}$.

For the small values of n discussed previously this reduces to the simple recursion relations of Eqs. (2) and (4). Specifically, for $n=2$ we have $(\lambda_1,\lambda_2)=(2,0)$ and $(0,1)$ as possible values so Eq. (16) becomes

$$H^2{}_m = [2;0,1]H^2{}_{m-1} + [2;2,0](H^1{}_{m-1})^2 \tag{17}$$

which is the same as Eq. (2) since $[2;0,1]=[2;2,0]=1$ and $H^1{}_m = 1$. Similarly for $n=3$ the possible values for the λ_i's are $(\lambda_1,\lambda_2,\lambda_3)=(3,0,0)$, $(1,1,0)$, and $(0,0,1)$ so

$$H^3{}_m = [3;0,0,1]H^3{}_{m-1} + [3;1,1,0]H^1{}_{m-1}H^2{}_{m-1} + [3;3,0,0](H^1{}_{m-1})^3 \tag{18}$$

which is the same as Eq. (4) because $[3;0,0,1]=1$, $[3;1,1,0]=3$, and $[3;3,0,0]=1$. Table I shows values of the $H^n{}_m$ numbers for various values of n and m.

We should note from this discussion that in the case of $m=2$, $H^n{}_2$, the number of ways to map n inputs to a single output in two steps, is just the number of ways of partitioning the set $\{1,2,\ldots,n\}$, into subsets. This is because after the first step any partition of $\{1,2,\ldots,n\}$ can be uniquely obtained by a single map and each of these must then map to the final output in the second step. Thus, when $m=2$, the $H^n{}_m$ reduce to the Bell numbers, which count the number of such partitions.[8]

Similar techniques can be used to derive a recursion relation for the value of Λ corresponding to a single basin of attraction that takes n inputs to a single output in m steps. In general, a map will consist of a collection of such trees and, from Eq. (9), the overall value of the tumbling index will be an average of the individual values weighted by the size of the various trees. To determine the recursion relation, one must know how the Λ values computed after $m-1$ steps can be combined to give the final value after m steps. We first consider a simple case in which two subtrees, t_1 and t_2, containing n_1 and n_2 inputs, respectively, are combined at the last step into a single tree t containing $n=n_1+n_2$ inputs. Thus the root node of t has two branches, one containing t_1 and the other t_2. Let Λ_1 and Λ_2 be the values of the tumbling indices

n=2 n=3 n=4 n=5

TABLE I. H numbers for selected values of n and m. Rows denote m, while the columns denote n.

A110

	1	1	1	1	1	1	1	
m=2	1	2	5	15	52	203	877	Bell nos
n=3	1	3	12	60	358	2471	19 302	258
	1	4	22	154	304	2915	146 115	~~[illegible]~~ 307
	1	5	35	315	3455	44 590	660 665	357
	1	6	51	561	7556	120 196	2 201 856	405
	1	7	70	910	14 532	274 778	5 995 892	1669
	1	8	dull	5945	new 5946			1765

TABLE II. Λ numbers for selected values of n and m. Rows denote m, while the columns denote n.

0.0	0.5	0.666 666	0.75	0.8	0.833 333 3	0.857 143
0.0	0.75	1.066 667	1.25	1.369 231	1.453 202	1.515 882
0.0	1.00	1.444 444	1.7125	1.894 972	2.028 194	2.130 06
0.0	1.25	1.818 182	2.167 208	2.409 202	2.588 85	2.728 291
0.0	1.50	2.190 476	2.619 048	2.919 132	3.143 922	3.319 863
0.0	1.75	2.562 092	3.069 519	3.426 999	3.696 282	3.908 137
0.0	2.00	2.933 334	3.519 231	3.933 719	4.247 126	4.494 55

for the subtrees. The problem is to determine the value of Λ, the tumbling index for the entire tree t. Using Eq. (9) and decomposing the sum over all inputs into sums over the two subtrees gives

$$\Lambda = n^{-2}(n_1^2\Lambda_1 + n_2^2\Lambda_2 + 2mn_1n_2) \tag{19}$$

This relation can be readily generalized to the case in which many equivalence classes are combined at the last step of the series of maps.

Now let $\Lambda^n{}_m$ be the average value of the tumbling index for all maps that take n inputs into a single output in m steps. As in the case of $H^n{}_m$, a recursion relation can be obtained by considering the typical partition shown in Fig. 7. The final value will be the average of the values obtained for each individual partition, weighted by the number of maps that produce that partition after $m-1$ steps. This factor is just the summand of Eq. (16). The Λ value for the map corresponding to the partition of Fig. 7 is

$$\Lambda_{\text{par}} = n^{-2}\sum_{i=1}^{n} \lambda_i[i^2\Lambda^i{}_{m-1} + m(n-i)i] \tag{20}$$

which depends on the particular partition through the values of the λ's. Note that in the case of a partition with just two sets this reduces to the expression given in Eq. (19). Averaging this result over all possible partitions then gives the final recursion relation for the tumbling index:

$$H^n{}_m\Lambda^n{}_m = \sum [n;\lambda_1,\lambda_2,\ldots,\lambda_n]\prod_{k=1}^{n}(H^k{}_{m-1})^{\lambda_k}\Lambda_{\text{par}} \tag{21}$$

with $\Lambda^n{}_1=(n-1)/n$. As in the case of Eq. (16), this can also be solved for $\Lambda^n{}_m$ provided the values of $\Lambda^k{}_m$, $k<n$, are known. For instance, when $n=2$ we have

$$\Lambda^2{}_m = (m+1)/4\ . \tag{22}$$

The tumbling indices for the trees of Fig. 2(a), in which two inputs are mapped together in three steps, are $\frac{1}{2}$, 1, and $\frac{3}{2}$, respectively. Their average is 1, which is just the value of $\Lambda^2{}_3$ given by Eq. (22). Values of $\Lambda^n{}_m$ for other indices n and m are shown in Table II.

III. EXTENSIONS OF THE THEORY

For the sake of simplicity, the previous discussion made a number of assumptions that will not hold in general. In this section we discuss how these can be relaxed to deal with constraints such as weighted errors, selective inputs, and restrictions on the kinds of possible maps.

(a) Weighted errors. In general, various errors are not equally likely to occur, so the averages used in Eqs. (6) and (8) should be weighted by the probability of a particular error occurring. For example, suppose the inputs are strings of bits of length k and each bit has an independent error probability of $\epsilon \le \frac{1}{2}$. Then the probability for the input i to produce an error i' which differs in j of the k possible positions is

$$w(i,i') = \epsilon^j(1-\epsilon)^{k-j}\ . \tag{23}$$

Note that the number of such i' will be given by the binomial coefficient

$$\begin{bmatrix} k \\ j \end{bmatrix} = \frac{k!}{j!(k-j)!}\ . \tag{24}$$

In this case, the Hamming distance between two inputs is an appropriate metric in that $w(i,i')$ depends only on the number of bits by which i and i' differ.

Returning to the example of balanced binary trees illustrated in Fig. 5 we can compute the values of P and Λ with this weighting. Since there are eight inputs in this case, they will be represented by strings containing three bits (from 000 for the first input to 111 for the eighth). The weighted version of Eq. (6) implies that $P(T,i)$ is the sum of the weights $w(i,i')$ for all inputs in the same tree as i. Thus in this example we have

$$P(T,1) = w(1,1) + w(1,2) + w(1,3) + w(1,4) = 1-\epsilon\ . \tag{25}$$

Note that when $\epsilon = \frac{1}{2}$, so that all weights are equal, $P(T,1)=\frac{1}{2}$ which is just its unweighted value. Similarly, for computing Λ, only inputs in the same tree are to be included. This means that the weights must be normalized so that their sum over all inputs in a particular tree is one. Thus Eq. (8) becomes

$$\Lambda(T,i) = \frac{\sum_{i'\in t_i} w(i,i')\Lambda(T,i,i')}{\sum_{i'\in t_i} w(i,i')}\ , \tag{26}$$

where the sums are over all inputs i' in the same tree as i. For this example we get

$$\Lambda(T,1)=[w(1,1)\times 0+w(1,2)\times 1 + w(1,3)\times 2+w(1,4)\times 2]/(1-\epsilon) = \epsilon(3-\epsilon) . \qquad (27)$$

Again, when $\epsilon=\frac{1}{2}$ this gives $\frac{5}{4}$ which is just the value obtained previously in the unweighted case.

(b) Selective inputs. In many practical instances only a small subset of all possible inputs are relevant for a particular application. This is particularly true when the input space is extremely large. Thus all maps that give the same tree structure for this special group, rather than for all inputs, can be considered equivalent. This effectively reduces the size, as well as the number, of the trees in that the other inputs can be ignored. However, they still play a role in determining the error-recovery rate because errors can still involve the other inputs. An example of such a system is given by common error-correcting codes, in which the desired outputs form a subset of the possible noisy outputs. Thus, the computation of the number of effectively distinct maps will only use the number of special inputs instead of the full value of n. The tumbling index, on the other hand, will still be determined by n, the total size of the input space. Note that the restricted set of inputs could have other special properties, e.g., being far from the boundaries of the basins of attraction, and therefore being more likely to recover from perturbations than one would expect from an average over *all* positions.

(c) Local maps. Previously, we have considered general maps in that any input could be mapped to any output independently of the others. In applications to computing structures that may be implemented as VLSI (very large scale integration) chips,[9] the issue of locality arises since it is easier to design circuits if the various elements do not require long wires to communicate with distant parts of the chip. It is thus desirable to consider only local maps. As we have recently shown, there is a class of locally connected architectures that can be made to compute in a distributed, self-repairing fashion, by exploiting the existence of attractors in their phase spaces.[2] More generally, locality will be required by any feasible computational process as the number of inputs becomes large. This is due to the fact that since the total number of general maps grows exponentially it is not feasible to specify arbitrary maps. For instance, if the data consists of k-bit (binary digit) integers (so that there are $n=2^k$ inputs) then a locality requirement means that a given bit of the output can only depend on a fixed, finite number of neighboring bits in the input. This greatly reduces the number of row-to-row maps that are possible and hence restricts the possible global behavior of the system.

As an extreme example consider one-bit maps, i.e., each bit in the output depends only on the corresponding bit of the input. In this case there are four possible local maps, i.e., ways to map $\{0,1\}$ onto itself, namely the identity map; inversion, in which $0\rightarrow 1$ and $1\rightarrow 0$; setting to 1, in which both 0 and 1 map to 1; and setting to 0. Since the actual output is not important for this discussion, there are only two distinct maps: either 0 and 1 can be mapped to different points (a noncontractive map) or to the same point (a contractive map). The local map that produces each bit of the output can be chosen independently, so that there will be a total of $2^k=n$ different maps. In the general, unrestricted, case there will be many more possible maps. Specifically, the number of distinct maps in one step is precisely the number of maps that produce a single output after two steps, i.e., $H^n{}_2$ or the nth Bell number.

More generally, for each bit of the output, the net effect of a series of m maps is determined by the location of the first contractive one-bit map. Since this can occur at any of the m steps or never occur at all, there will be $m+1$ possibilities for each bit. Thus there are $(m+1)^k$ different m-step maps consisting of one-bit local maps. If all of the inputs are to be in a single basin of attraction, then there must be at least one contractive map in the series for each bit. Thus the total number of local one-bit maps that take $n=2^k$ inputs into a single output in m steps is m^k compared to $H^n{}_m$ in the general case. For example, if $k=2$ and $m=3$ then $m^k=9$ whereas $H^n{}_m=H^4{}_3=60$. The restriction to local maps can also be viewed as a constraint on the possible partitions of the set of inputs $\{1,2,\ldots,n\}$ that can be realized as equivalence classes for the maps. In this case, for instance, the realizable partitions will consist of equal-sized subsets.

As illustrated by this example, restricting the kinds of maps that can be used reduces the range of behavior that the overall system can exhibit. This has the practical consequence of creating situations in which constraints on the system cannot always be satisfied by the restricted set of maps.[10]

(d) Long-time behavior. An additional issue is posed by the case of very-long-time behavior, i.e., when m is not much smaller than n. Instead of focusing only on the final output after a fixed number of steps, it is then useful to examine the behavior of the maps as a function of m. If the *same* map is used at each step, then the behavior will consist of a transient of average length $n^{1/2}$ followed by a fixed point or a cycle.[11] On the other hand, if the maps differ arbitrarily from step to step, even longer transients are possible and the system may never settle into a final cycle at all.

IV. EXPERIMENTAL CONNECTIONS WITH COMPUTING STRUCTURES

In this section we indicate how the theory developed in this paper can be applied to real machines that implement discrete dynamical systems.

(a) Maps from computational rules. A major application of these ideas is to computing structures. In this case the individual maps are implemented by a set of computational rules in either hardware or software. Since the total number of possible maps grows very rapidly with the number of inputs, those that can actually be implemented will be a restricted subset, e.g., the machine will only be capable of providing local maps. A proper analysis of the reliability and other characteristics of such computing structures requires a combination of a number of the factors discussed in Sec. III. For instance, the actual error probabilities for the system will determine the weighting function to use, and generally only a small subset of all

ssible inputs will be important. For these finite-state chines, quantities such as P and Λ will indicate their iability. For conventional computing circuits, small nsi [illegible] or permanent errors, in their internal states or a, often lead to major global changes in behavior.[12] her kinds of machines, exploiting the collective avior of many parallel processors, can be made largely mune to such perturbations by using the existence of attors.[2]

The particular example that we have studied is a chine composed of a rectangular array of simple prosors each of which operates on integer data received loly from its neighbors. Overall input and output to the chine takes place only along the edges. Each processor an internal state or bias, represented by an integer, ich can only take on a small set of values. The unit ermines its local output based on its inputs and its ernal state. At each time step, every element receives a values from the units to its upper left and right, pectively, and computes its output according to a simrule. This rule leads in turn to a contraction mecham whereby many inputs are mapped into the same out. In the language of dynamical systems, this corrends to the appearance of a fixed point in the phase ce of the system. Furthermore, the contraction of umes in phase space makes these fixed points attractive the sense that perturbations quickly relax back to the ginal values. This machine is capable of recovering m errors in either the data or its internal state, and is to produce reliable pattern classification.

We [illegible] found that adding an adaptive process which ws the maps themselves to change depending on the uts and a specification of the required response can n produce the desired classifications. Note that this ptive process amounts to modifying the maps as a ction of time. In our machine, this process usually verges quickly to a fixed point at which the maps no ger change. At this point, the quantities we have ded for a static series of maps can be evaluated. For a ow—by—6-column array with 64 attractors, a meament of Λ using techniques described below gave an age value of 4.2. This number implies that on the age errors recover in 4 steps, which is less than 10, the th of the array. In this way, one can investigate the le-offs between increased reliability and the ability to sfy other external constraints such as producing a ified grouping of patterns. Additional details conning the architecture and behavior of this machine are n in Ref. 2.

b) Measurement techniques. Determining the particutree that arises from a given set of computational s, and hence deducing its characteristics analytically, ifficult for all but the simplest rules. There is, howevan experimental approach that can be used to estimate parameters that we have defined in this paper for a set ules. Specifically, one can use random samplings or aust[illegible] searches to estimate the number and sizes of va[illegible] basins of attraction. Furthermore, for those iples that do reproduce the original output, one can ord the corresponding healing length and hence estie the tumbling index or ultrametric distance. By weighting the selection of samples by the likelihood of errors, the correct values of P and Λ can then be obtained. These quantities can in turn be used to estimate the reliability of the system. We should point out that although some architectures have a treelike layout, it is not possible to make an *a priori* connection between the architecture and the tree corresponding to the map that it implements.

V. CONCLUSION

In this paper, we have developed a theory that provides a natural language for describing the global behavior of discrete dynamical systems which are both finite and dissipative. By considering the combinatorial properties of discrete maps, we were able to obtain quantitative definitions of the parameters relevant to the dynamics of such systems. Thus, unlike continuous dynamical systems, attractors on finite sets produce behavior characterized by exact recovery from fluctuations in a finite amount of time. This result has a direct implication for the time evolution of computing structures and can also be applied to other complex systems with hierarchical structures.

An important extension of this theory relates to adaptation and time-dependent maps, particularly if the change depends on the time sequence of inputs. In this way, the maps become active participants in the dynamics and provide a much richer set of possible behaviors. This variation could be due to loops in which the output of a particular step forms part of the next input to that step, or, more generally, the form of the map could depend on the past history of all inputs. This possibility occurs in adaptive systems of the type that we studied earlier. In that case, one is interested in studying the reliability of such computing structures against perturbations in either data or the maps themselves. This paper suggests a crisp methodology for doing so. For example, one can determine the stability of the system to perturbations that change the maps by measuring quantities such as the tumbling index. Also, a determination of the variability in the branching ratio of the tree produced by such structures would give an indication of their degree of complexity. Finally, since real computers operate in a world with constraints, our calculations provide an estimate of the trade-off between reliability and capacity to satisfy those constraints.

ACKNOWLEDGMENTS

We benefited from useful comments by P. Stein. This research was partially supported by U.S. Office of Navy Research (ONR) Contract No. N00014-82-0699.

APPENDIX

In this appendix we derive generating functions for the H numbers from their recursion relations. First, we note that the generating function for the modified multinomials $[n;\lambda_1,\lambda_2,\ldots,\lambda_n]$ is[7]

$$\frac{1}{m!}\left[\sum_{k=1}^{\infty}\frac{x_k t^k}{k!}\right]^m = \sum_{n=m}^{\infty}\frac{t^n}{n!}\sum_{\lambda_1,\lambda_2,\ldots,\lambda_n}[n;\lambda_1,\lambda_2,\ldots,\lambda_n] \times x_1^{\lambda_1}x_2^{\lambda_2}\times\cdots\times x_n^{\lambda_n}\,, \tag{A1}$$

where the sum over λ_i's is restricted to values satisfying both $\sum_{i=1}^{n} i\lambda_i = n$ and $\sum_{i=1}^{n}\lambda_i = m$. The recursion relation for $H^n{}_m$, Eq. (16), involves a sum in which the λ's are only restricted to values satisfying $\sum_{i=1}^{n} i\lambda_i = n$. Thus it is necessary to sum Eq. (A1) from $m=1$ to ∞. Using Eq. (16) we then have

$$S_m(t)=\exp[S_{m-1}(t)]-1\,, \tag{A2}$$

where we have defined the generating function

$$S_m(t)=\sum_{k=1}^{\infty} t^k H^k{}_m/k!\,. \tag{A3}$$

Note that $H^k{}_m$ can be obtained from $S_m(t)$ by differentiation, i.e.,

$$H^k{}_m = \left.\frac{d^k}{dt^k}S_m(t)\right|_{t=0} \tag{A4}$$

and $S_0(t)=t$ since $H^k{}_0=\delta_{k1}$. From Eq. (A2) we see that $S_m(t)$ can be expressed as a series of iterated exponentials. For example, $S_2(t)=\exp(e^t-1)-1$ which is the generating function for the Bell numbers.[8]

Finally, we should point out that a similar set of functions can be derived for the tumbling indices $\Lambda^n{}_m$ based on the recursion relation given in Eq. (21).

---

[1]For an illustration of this methodology, see, for example, V. I. Arnold, *Mathematical Methods of Classical Mechanics* (Springer, New York, 1978).

[2]B. A. Huberman and T. Hogg, Phys. Rev. Lett. **52**, 1048 (1984); T. Hogg and B. A. Huberman, J. Stat. Phys. (to be published).

[3]B. A. Huberman and M. Kerszberg, J. Phys. A **18**, L331 (1985).

[4]See, for instance, *Kinetic Logic*, edited by R. Thomas (Springer, Berlin, 1979).

[5]N. Bourbaki, *Espaces Vectoriels Topologiques* (Herman, Paris, 1966).

[6]See, for instance, R. G. Gallager, *Information Theory and Reliable Communication* (Wiley, New York, 1968).

[7]M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions* (Dover, New York, 1965).

[8]For a general discussion of partitions, see J. Riordan, *An Introduction to Combinatorial Analysis* (Wiley, New York, 1958).

[9]L. Uhr, *Algorithm-Structured Computer Arrays and Networks* (Academic, New York, 1984).

[10]A typical example is given by the dynamic frustration effect reported in Ref. 2. In this case, a program which tries to sequentially modify the basins of attraction of an adaptive computing structure never halts when successive instructions undo the effects of the previous ones.

[11]D. E. Knuth, *The Art of Computer Programming* (Addison-Wesley, Reading, Mass., 1973), Vol. 1. Also, J. Coste and M. Henon (unpublished).

[12]Error mechanisms in actual microelectronic circuits are discussed by J. McNuthy, Phys. Today **36** (1), 9 (1983).