ON DESIGN PRINCIPLES FOR A MOLECULAR COMPUTER

If the unique information-processing capabilities of protein enzymes could be adapted for computers, then evolvable, more efficient systems for such applications as pattern recognition and process control are in principle possible.

MICHAEL CONRAD

It is only recently that a serious interest in the possibility of using carbon-based materials for computing has developed in the scientific community. Formerly, individuals expressing an interest in exploring this possibility would have been advised to implement their computing concepts in existing electronic technologies. Indeed, to the best of my knowledge, no molecular computing device has so far been constructed or shown to be imminent. However, a convergence of developments in a number of fields, including polymer chemistry, various biotechnologies, the physics of computation, and computer science, has changed the situation. Theoretical arguments suggest that more efficient and adaptable modes of computing are possible, while emerging biotechnologies point to possibilities for implementation. Their common ground is molecular computing.

One immediate objective is to produce a von Neumann-type computer in carbon, rather than silicon [3]. The assumption is that carbon chemistry may facilitate the construction of smaller and faster devices. Attention has therefore been focused on the possibilities for organic switching devices and conducting polymers [21, 41], and secondarily on the problems of contact and reliability. Certainly, work in this area is potentially important. Even so, it is likely that molecular computing will prove to be much more valuable outside the context of conventional von Neumann computers. Critically important computing needs such as adaptive pattern recognition and process control may be refractory to simple decreases in size and increases in speed. Instead of suppressing the unique properties of carbon polymers, we should consider how to harness them to fill these needs.

The information-processing capabilities of biological systems, which are ultimately based on the conformational properties of protein enzymes, suggest this approach (see Figures 1–3). Although elaborate conformation is probably incompatible with good conductivity, it allows for the lock-and-key type interactions that enable enzymes to recognize specific molecular objects by exploring their shapes.[1] Shape-based specificity is a form of tactile pattern recognition. Admittedly, enzymes make for much slower switches than transistors (typically 0.1 millisecond, as compared to a nanosecond). To simulate the sensory and control functions they perform, however, would require an enormous number of switching processes in a digital computer. Designs that use functions of this sort as *primitives*—predefined elements that are irreducible as far as the computing power of the machine is concerned—can reasonably be expected to yield significant increases in computational power for such tasks as pattern recognition and process control. In addition, enzymes have the virtue of being adaptable switches, which makes them amenable to tailoring for particular functions through trial-and-error evolution. The evolution process proceeds by variation of the sequence of amino acids in the enzyme, followed by selection and propagation of the best-performing sequences (see Figure 4).

However, these advantages can only be exploited at the expense of programmability, which is the major feature of today's computers. This idea can be stated in the form of a trade-off principle: *A system cannot at the same time be effectively programmable, amenable to evolution by variation and selection, and computationally efficient.* The von Neumann computer opts for programmability. The trade-off theorem suggests that an alterna-

[1] The argument is that the conformational flexibility of biopolymers is concomitant to the lack of conjugation required for electronic conductivity [43].

tive domain of computing is in principle possible, where programmability is exchanged for efficiency and adaptability. Biological systems, as the products of evolution, must operate in this alternative domain.

ESSENTIAL FEATURES OF VON NEUMANN COMPUTERS

Let us first list some of the fundamental features of von Neumann computers. These features have become so familiar that it is possible to forget how remarkable they really are [6]:

1. *Programming languages exist.* It is a fact of experience that if we can conceive of an algorithm we can always express it directly in any of a large number of computer languages (e.g., ALGOL, LISP, Pascal). A basic feature of all such languages is that they are defined by a finite, discrete base set of symbolic primitives. Were these symbols not finite in number, a user's manual could not be finite either. Were the symbols not discrete, it would be necessary to perform calculations to ascertain what algorithm a program actually expresses. The psychological sense of directness conveyed by today's computer languages is derived from these properties.

2. *Effective programmability.* Once a program is written, it is always possible to effectively communicate it to an actual machine. From a user's point of view, this communication process may be mediated by an interpreter, which can read and follow any particular program, or by a compiler, which sets the state of the machine so as to execute the desired program.

3. *Structural programmability.* Digital computers all employ a small repertoire of simple switches (e.g., logic gates and on-off devices). Each switch constitutes a switching primitive. It is always possible to write a program in a language that directly maps the structure of the machine and the state of each component [8, 14]. This property, *structural programmability*, is the basis for all other forms of programmability in today's computers. The network in Figure 5 illustrates the concept of structural programmability. It shows how algorithms expressed in terms of the symbolic primitives of a highly simplified programming language can be expressed directly in terms of the switching primitives of the network. Of course, the diagram of the network is also a language, and the "switching" primitives in it are also symbolic primitives. However, they are symbolic primitives that can easily be realized as switches.

4. *Universality.* Were there no limitations on time and space, the von Neumann computer could run any conceivable program. It is also universal in that, so far as is known, it can simulate any physically realizable process in nature (more on this in the next section).

5. *Sequentiality.* It is possible for the von Neumann computer to execute a single elementary operation at a time. True parallelism, as where two or more programs have access to the same computational resources at the same time, is also possible, but, in general, only at the expense of effective programmability (for a discussion of parallelism and speedup, see [29]).

The von Neumann computer also has fundamental shortcomings:

6. *Inefficiency.* Von Neumann computers make inefficient use of available space, time, and energy resources. The vast majority of processors are dormant at any given time.

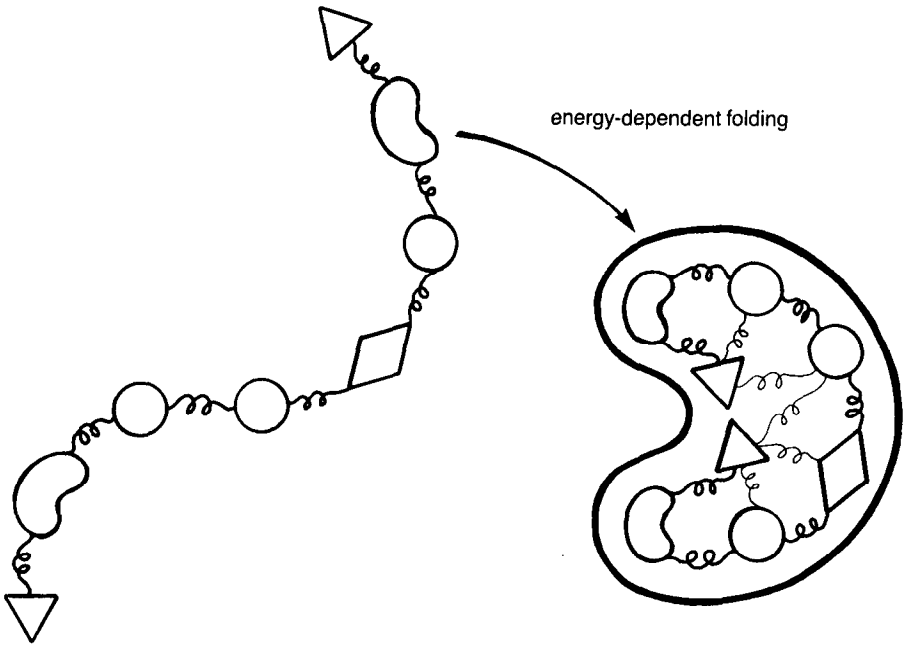
7. *Sensitivity.* The sensitivity of programs to change is a matter of common and usually unpleasant experience.

The strengths of the von Neumann design (properties 1–5) are all control properties. Intensive efforts are under way in computer science to retain these control properties while removing the constraints on efficiency and adaptability (properties 6 and 7). The trade-off principle suggests that the strengths of the von Neumann computer are inextricably linked to its limitations.

THE TURING–CHURCH THESIS

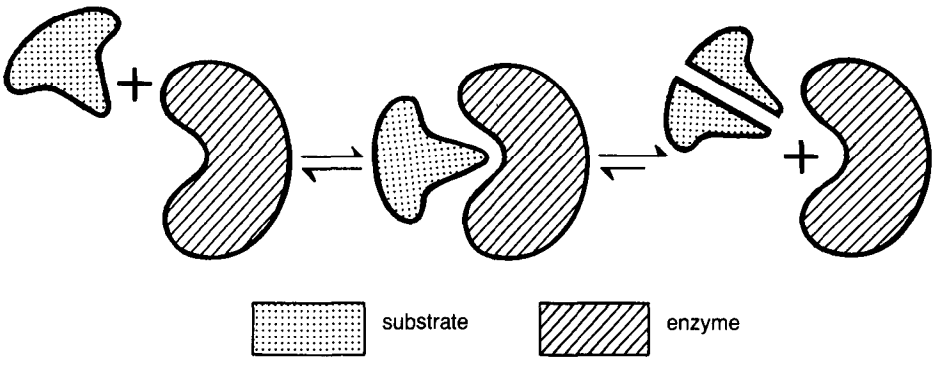
The capabilities computer scientists generally associate with the von Neumann computer are summed up in the famous thesis of Turing and Church (a recent general discussion can be found in [24]). One statement of this thesis is *any effectively computable function is computable by a formal process involving simple operations on strings of symbols.* A program is a rule that generates such a process. The classic Turing formalization of a computing process is illustrated in Figure 6. The Turing machine program can be rewritten in any general-purpose algorithmic language. These various formalizations all define the same class of functions—the partial recursive functions. To prove this, it is only necessary to show that they are all equivalent up to the point of simulation, that is, up to the coding of strings of symbols into other strings of symbols. This equivalence supports the Turing–Church thesis, although it does not actually prove it. Indeed, proof is impossible, since the claim is that an *informal* concept of computation is equivalent to any of a large number of *formal* models of computation. Disproof, however, is possible if a convincing counterexample can be found. Computer scientists generally accept Turing–Church because no such counterexample has ever been discovered.

The Turing–Church thesis is sometimes interpreted narrowly as encompassing only the processes of logic and mathematics. The strongest interpretation is that *any physically realizable system or process must be effectively computable.* If a physically realizable system or process were not effectively computable, it could be used as a new primitive of computation, thereby enlarging the class of functions computable by systems in nature. For the purpose of this article, I accept this strong form of the Turing–Church thesis, first because



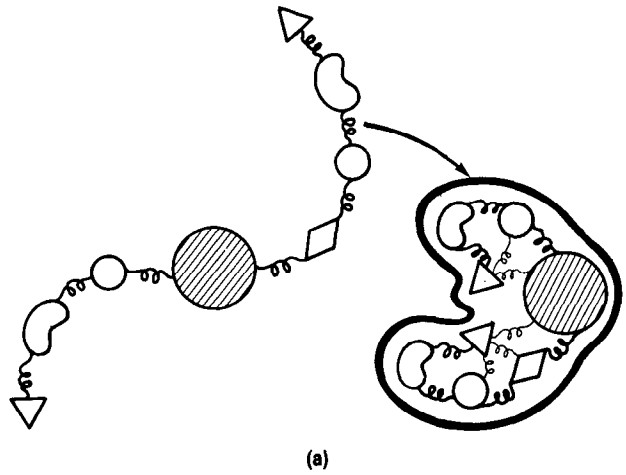
The sequence of nucleotide bases in DNA, which stores the information accumulated during the course of evolution, is translated to the sequence of amino acids in proteins. The amino-acid sequence is pictured schematically as a string of beads, each different type of bead standing for one type of amino acid. Weak interactions among the amino acids cause this linear representation of information to spontaneously fold up, forming a three-dimensional spatial structure. Strong or covalent bonds are pictured with dark springs, and weak bonds, which determine the folded structure, by light springs. Only some features of the enzyme's folded shape are critical for its function. Here, the relative position of the triangles conveys the critical shape feature. Natural proteins are built from 20 types of amino acids. A typical protein might comprise a folded string of 300 amino acids. The functions performed by the protein are determined by its three-dimensional structure and dynamics.

FIGURE 1. The Representation of Information in Proteins

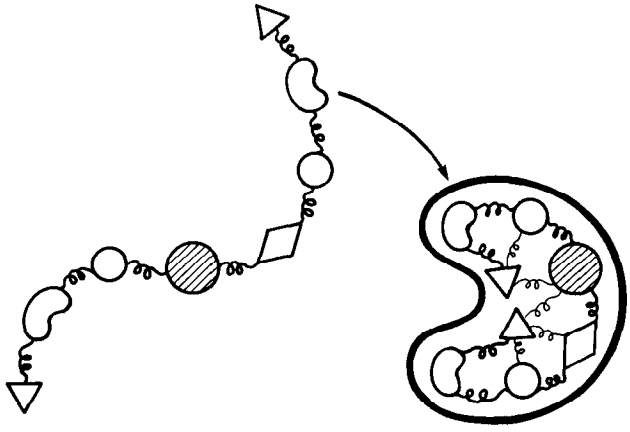


A protein serves as an enzyme, or biological catalyst, if it speeds the formation or severing of a particular covalent bond in a particular target molecule, called the substrate. In so doing, the protein *switches* the substrate from one state to another. Before switching can occur, the enzyme must recognize the substrate—that is, distinguish it from other, possibly quite similar molecules. Recognition is a tactile phenomenon involving the complementary fit of enzyme and substrate shapes. The enzyme and substrate explore one another by means of diffusional motion. When a close fit occurs, short-range interactions between the two molecules become significant. Shape-dependent recognition is sometimes described as a "lock-key" mechanism; in reality, though, the dynamical properties of the enzyme also play a role. The enzyme may assume different shapes, or states with different functional properties, as a result of interactions with control molecules. Features of shape may serve as binding sites that allow the enzyme to recognize and stick to specific molecular structures. Some proteins with purely structural functions use this type of lock-key interaction to self-assemble into larger molecular structures.

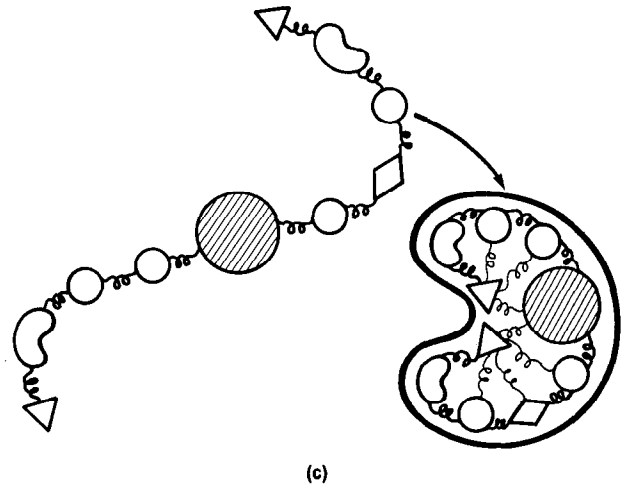
FIGURE 2. Proteins as Pattern-Recognizing Switches



(a)



(b)



(c)

New proteins arise through variation mechanisms, such as mutation, which alter the amino-acid sequence. A single mutation of a crucial amino acid may disrupt the ability of the protein to recognize and switch substrates. Other mutations produce only slight changes in shape and function. Diagram (a) shows how folding can serve to distribute the effects of mutation over the whole structure. The mutated amino acid is represented by the large striped circle. The relative orientation of the triangles, which represent a feature of shape critical for function, is altered as compared to the unmutated protein illustrated in Figure 1. By distributing the effects of mutation, folding allows the critical shape feature (i.e., the orientation of the triangles) to change only slightly in a significant number of cases. The effects of mutation on this feature may be further buffered by redundancies: Redundancy in the number of types of amino acids allows for mutation to closer structural analogs, as represented by the somewhat smaller striped circle in (b). Redundancy in the number of amino acids increases the capacity of the protein to absorb the effects of mutation in features of shape that are not critical for function. This is illustrated by the partially restored alignment of the triangles in (c). Redundancies that buffer the effects of mutation lead to an enormous speedup of the evolution process.

FIGURE 3. The Evolutionary Adaptability of Proteins

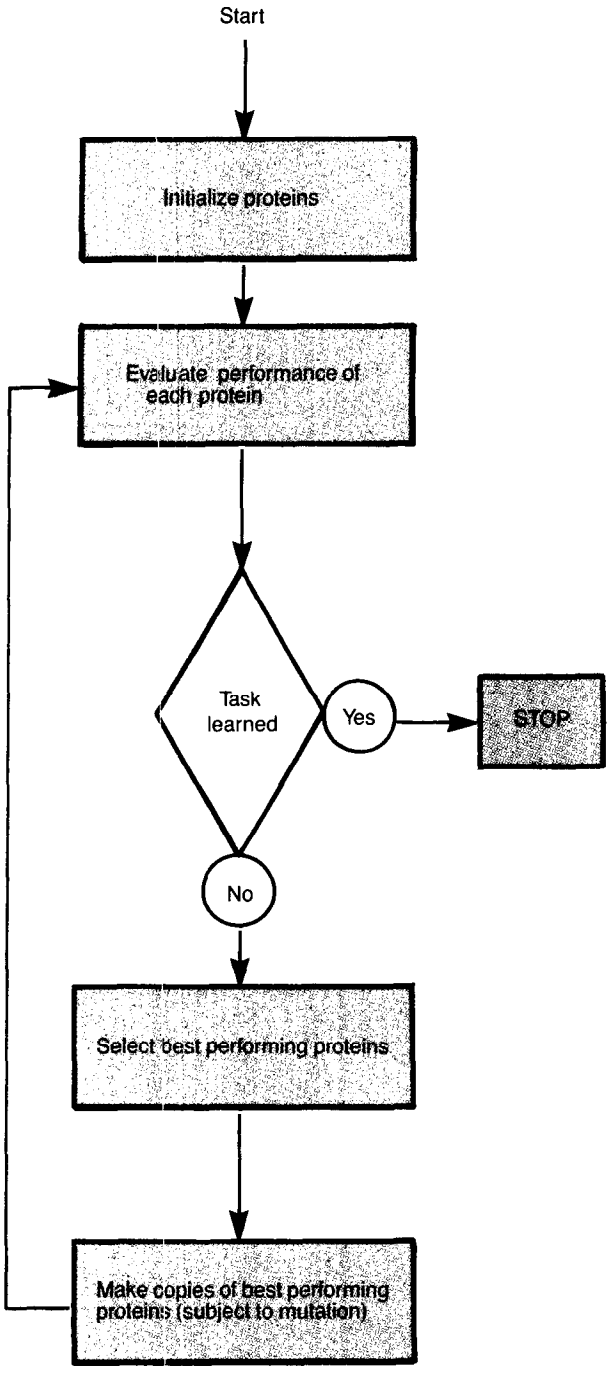


FIGURE 4. The Evolutionary Learning Algorithm

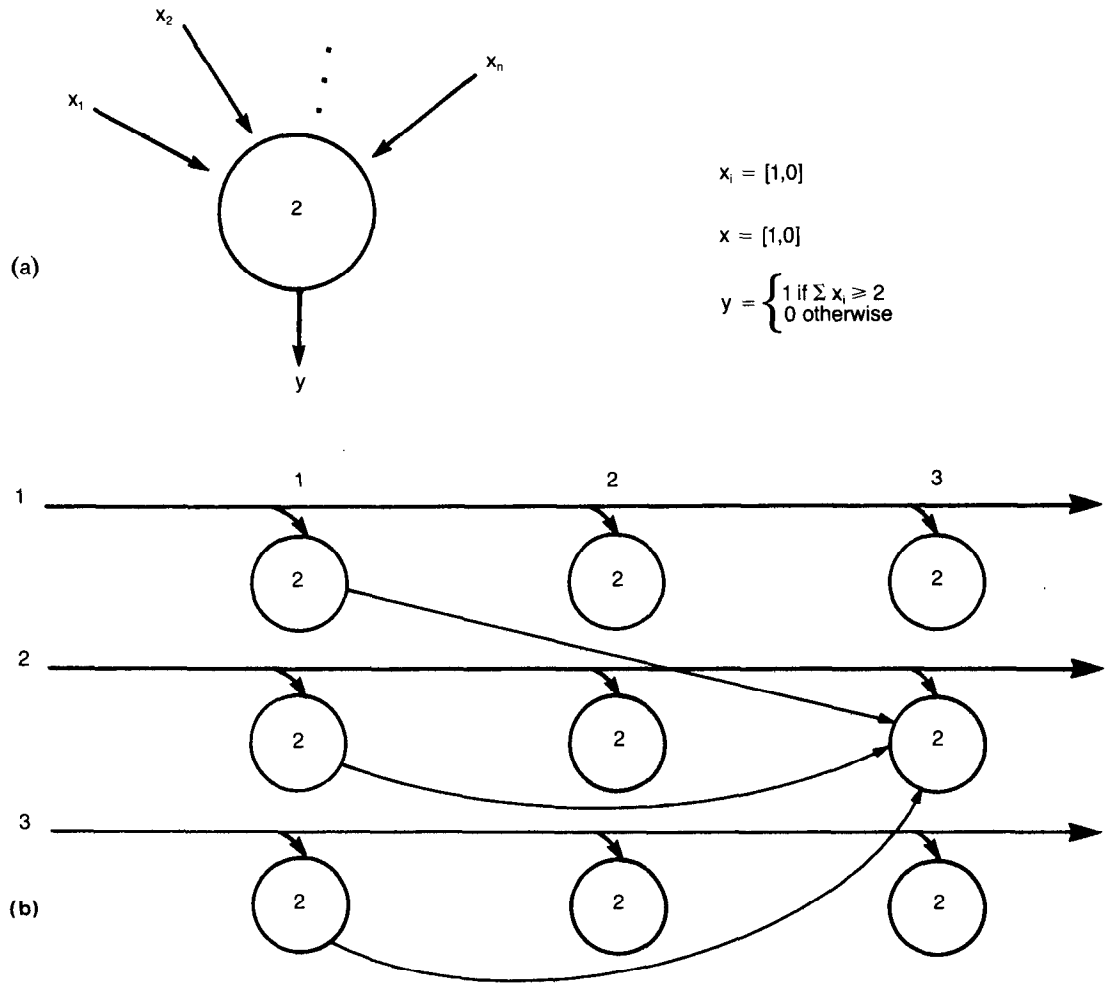
The "programmer" specifies the task and the criteria for success. Evolution proceeds by variation of the amino-acid sequences of the proteins. Selection is based on how well each protein performs a desired task. In general this evaluation is based on the performance of the system in which the protein is acting, just as in nature it is based on how well the organism performs. Better-performing systems are said to be more "fit." Propagation of fit proteins is achieved by producing copies of them rather than of less fit proteins. Improvement is achieved by coupling the copy process to the variation process. In nature this is achieved by differential reproduction of organisms, where reproduction is coupled to variation of genes. The artificial selection process pictured here differs in two major ways from evolutionary processes in nature. First, there is the fact that in nature many genes control the traits of organisms. Higher mechanisms of variation such as crossing over, recombination, and variations affecting the regulation of gene expression are important. As with simple mutations, however, these higher mechanisms require the organizations on which they act to be capable of improvement through sequences of single variation events in order for them to make an effective contribution to the evolutionary process. The second major difference is that the task-definition and selection processes are artificial, that is, determined by a programmer, rather than consequences of ongoing interactions among organisms or between organisms and their environment. The evolutionary programmer specifies goals in terms of criteria of selection, whereas a digital-computer programmer specifies structures to achieve the desired goals.

what is of interest is the computational uses to which physically realizable systems can be put, and second because the thesis would be rather trivial if physically realizable systems existed that could violate it.

THE EQUIVALENCE OF DYNAMICS AND COMPUTATION

Taken in this strong form, the Turing–Church thesis provides a link between formal models of computation and dynamical processes far removed from those conjured up by the mechanisms present in a von Neumann computer. The von Neumann computer is a physical realization of a formal system executing simple operations on strings of symbols. We say that such a computer *simulates* a dynamical system if states of the computer can be made to correspond to states of the dynamical system at each point in time to an arbitrarily high degree of approximation. (We do not, however, require that every state of the machine correspond to a state of the dynamical system [15].) According to Turing–Church, all physically realizable dynamics *are equivalent to computation* in that they can be simulated by a von Neumann computer under the idealization that space and time bounds can be ignored. If we could demonstrate that the dynamics of a particular physical system could not be so simulated, then we could use this system to solve problems effectively (i.e., by a definite procedure) that would be unsolvable by a digital process. For example, we could solve the problem of generating the behavior of this particular dynamical system. Either the Turing–Church thesis implies that all physically realizable processes can be duplicated by digital computers, or it fails to put any limits on what real systems are effectively capable of doing.

Of course, the simulation of a phenomenon—of a nuclear explosion, for instance—is not the same thing as the phenomenon itself. All simulations are abstract in that they fail to capture some aspect of the simulated system. This is the case whether the simulation is of

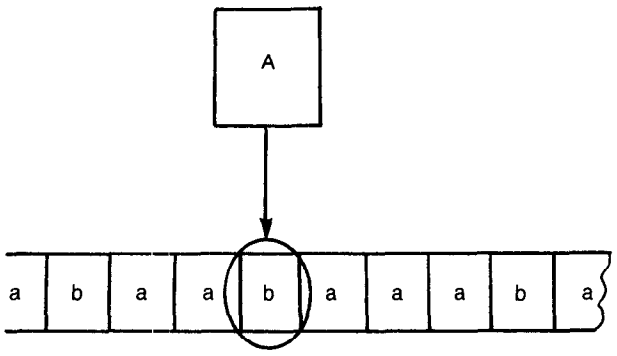


For simplicity we consider a machine (technically a semi-automaton) with three inputs, x_1, x_2, and x_3, and three states, q_1, q_2, and q_3. The program is a sequence of triples of the form $q_i x_j q_k$. The primitives of the construction are McCulloch–Pitts formal neurons (a). These fire if the sum of their inputs (inputs can be 0 or 1) equals or exceeds a threshold, which in this case is always 2. In the canonical network (b), state q_i is coded by the firing of a formal neuron in column i, and input x_j by the firing of input line j. The programmer's manual consists of a single rule: *To implement $q_i x_j q_k$, draw a connection between all neurons in column i and the neuron in row j of column k.* For clarity the only instruction implemented is $q_1 x_2 q_3$. The example shows that it is possible to express programs directly in the network structure, though it may be more convenient to do so in terms of the triples $q_i x_j q_k$. The construction was suggested by a somewhat different one by Minsky [35].

FIGURE 5. The Concept of Structural Programmability

one string process by another or of a continuous dynamical process by a string process, as in the example of the nuclear explosion. The execution sequence of the computer has the same ontological relationship to the real explosion as do the hydrodynamic equations that describe the explosion. The equations are a map from states to states that is meaningfully defined only in terms of some method of computation. The explosion and the machine that simulates it can be thought of as alternative means of computing this map. The real content of the Turing–Church thesis, if it is correct, is that any physically realizable map can also be realized by the execution sequence of a physical system performing formal operations on strings of symbols. Many maps can be imagined that are not computable by simple operations on strings of symbols—a map of all computer programs into the two categories, correct and incorrect, for instance. Computing this map is equivalent to solving the halting problem, which we know to be unsolvable by a Turing machine or any process equivalent to a Turing machine [16]. If an actual system could realize this map, it would constitute a new primitive that could not be duplicated with any set of primitives formerly believed to define the class of computable functions.

The proposed equivalence of dynamics and computation makes it clear that tying the concept of computing to any single model of computation is arbitrary. Any physically realizable process whatsoever can be admit-



The three components are a finite-state machine (A), a read-write head, and a tape that can be read, marked, or moved a square to the right or left. The inputs to A are tape symbols, and the outputs either tape symbols or moves. The program of the Turing machine can be expressed as a set of quadruples of the form $q_i x_j y_k q_l$, where q_i is the initial state, x_j the symbol read on the tape, y_k the output symbol or move, and q_l the final state. Turing thought of A as an abstraction of a person performing some symbolic process (e.g., an arithmetic process) on a notepad (i.e., the tape). The program expresses the rule according to which the "states of mind" of A change. The finite-state device can always be realized by a set of McCulloch–Pitts formal neurons of the type shown in Figure 5.

FIGURE 6. A Turing Machine

ted as a computational primitive. Our current preference for simple switching processes can be attributed to our desire to exert prescriptive control over computation. Even if a dynamical process is effectively computable, the machine that incorporates this process as a primitive need not be effectively programmable. The discrete, finite aspect of the primitives would be lost, and with it the ability to communicate algorithms directly to the machine. (Analog computers are of course an exception of sorts. It is possible to use a precisely defined system analogy to guide design. Admitting arbitrary physical processes as primitives of computation is generally incompatible with prescriptive design, however.)

We have observed that all simulations are abstract in that they only *represent* what they simulate. This raises the question, could this "abstractness" be a significant limitation? The philosopher Ludwig Wittgenstein argued that the interpretation of a rule could never be an intrinsic property of the rule itself [48]. For example, the interpretation of the symbol "→" is not inherent in that symbol. If we accept the idea that all essential aspects of human intelligence can be duplicated by following formal rules expressed as computer programs (i.e., by following algorithms), then we must maintain either that rules can carry their own interpretation or that an interpretation can be an emergent property of a set of rules. If symbol interpretation cannot be achieved in these ways, it could be argued that machines like the von Neumann computer, which are constrained to realize formal sets of rules, are limited in their ability to duplicate a crucial aspect of human intelligence. Such a limitation would not necessarily apply to systems like the molecular computer design here described, which are not so constrained. For ascertaining whether it is useful to enlarge the pantheon of computational primitives, however, it is sufficient to accept the Turing–Church thesis as formulated. The class of Turing computable functions is so much larger than the class of functions that could be computed by any finite system that it is the comparative performance within this class, indeed within a rather minute subdivision of it, that is the issue of immediate concern. It is sufficient to show that a molecular computer could perform some useful tasks that could only be performed by a present-day computer at an excessive cost in terms of computational resources.

PROGRAMMABILITY VERSUS EVOLVABILITY

The first part of the trade-off theorem states that programmability is a cost in terms of evolutionary adaptability. Recall (from Figure 5) that a system is structurally programmable if it is possible to communicate algorithms to it by setting the states and connections of its physical primitives using a finite programmer's manual. Consider two structurally programmable computers P and P', which differ by a single structural feature (corresponding to a single alteration according to the construction manual). Given the same input, how different will the processes executed by these two systems be? This problem, which I call the *unsolvable transformability problem*, is unsolvable in just the sense and for just the reason that the halting problem for Turing machines is unsolvable [9].

Suppose, as the reductio hypothesis, that it is possible to write a program U to solve the gradual transformability problem. This program could use any nontrivial metric to measure the similarity of the two processes. By a nontrivial metric, I mean any metric that does not result in P and P' being classified as similar independent of what states they are in. Thus it is only necessary to divide the state set of P and P' into at least two disjoint, nonempty subsets of states. Suppose that the criterion of similarity is that, given the same input, P' gives a defined computation whenever P does. The computation is defined if the machine running the program reaches a halt state. We are free to take as P any program that halts. For the reductio hypothesis to hold, it is at least necessary that U go to a halt state and emit as output "not similar" whenever P' does not go to a halt state. Suppose that U is itself P'. Then U halts if it does not halt, that is, answers "not similar" when it does not answer "not similar." Since this is clearly a contradiction, the assumption that U is a possible machine must be incorrect.

The unsolvability of the gradual transformability problem corresponds to the common experience that a

single change in a computer program (other than a parameter change) usually leads to major changes in the execution sequence. Rarely does such a randomly altered system serve any useful function. Redundancies can be introduced to confer fault tolerance, but in this case changes in function are actually prevented. Fragile systems of this type are unsuited to evolution; variation and selection is efficient as a method of self-organization only if a useful P' can always be produced from P by a single structural alteration. The probability of an improvement or of a step that can bring the system closer to an improved form is then proportional to p, the probability of a single change. If n simultaneous changes are necessary, the probability of an improvement is proportional to p^n, a very small number for any reasonable choice of p even when n is 2 [5, 12]. Large values of p are never reasonable, since these always lead to the introduction of many undesirable changes, except in trivially small programs.

This argument is the basis for the first part of the trade-off principle: that evolution is not compatible with structural programmability. This is not to say that evolutionary processes cannot be simulated on structurally programmable computers. After all, according to Turing–Church, it must be possible to simulate evolution, otherwise a physically realizable process would exist in nature that is not effectively computable. Computer programs have in fact been written that successfully use the evolutionary principle (e.g., Samuel's checkers program [40], Bremermann's optimization algorithm [2], and evolutionary learning algorithms used in simulations of structurally nonprogrammable systems [25]). The key to all such programs is restricting evolutionary changes to parameters and parameterizing problems as much as possible. The logical extension of this strategy is to increase the degree of parameterization by simulating the types of dynamical features and redundancies that facilitate the evolutionary process in biological systems. Continuous dynamical features are evolution facilitating since they make it possible to define a small perturbation in a way that would be impossible in a formal system. For instance, a structurally stable developmental process can undergo a wide variety of useful transformations in response to genetic change, while retaining coherent organization [38, 44]. Redundancies are evolution facilitating since they can buffer the effects of genetic change on features critical for function [11]. To the extent that we are willing to pay to simulate such evolution-facilitating features, we can build virtual machines having an effective evolutionary capability on top of von Neumann machines. The underlying program would remain fragile, however, and the cost of simulation would have to be weighed against the potential benefits.

The simplest and most important illustration of a system structured for effective evolution is the protein enzyme [13]. The enzyme assumes its shape and function through an energy-dependent folding process, which in a significant number of cases is only slightly altered by a point mutation—that is, by a single change in amino-acid sequence [45] (See Figure 3). Portions of the enzyme that are critical for function, like the active and control sites, can be buffered from the effects of point mutation by functionally less significant portions of the molecule. "Programming" occurs at the level of the amino-acid sequence, but structural programmability is lost since shape and function emerge from this sequence through a continuous dynamical folding process. The intervention of this continuous dynamical process increases the likelihood that single mutations will lead to functionally acceptable forms of the enzyme, and is thus critically important for maintaining a nonnegligible rate of evolution. As we have seen, this is because the evolutionary rate decreases sharply when it depends on the simultaneous occurrence of two or more structural variations. Assuming that it someday becomes possible to compute the shape and function of proteins from primary structure, we might entertain the idea of building a virtual machine with open-ended evolutionary capabilities using the von Neumann computer as the base machine. The computational costs of simulating folding processes would, however, be prohibitively high—much higher than the cost of simulating features that would confer a moderate evolutionary capability.

PROGRAMMABILITY VERSUS EFFICIENCY

We have seen that simulating evolution on structurally programmable computers is computationally expensive. In this section we show that structural programmability can be exchanged for computational power. To see why this is so, let us consider the operation of a structurally programmable computer at three levels.

The network level. The efficiency with which the computational resources of a computer are utilized increases as a greater fraction of processors in the system is active at any given time. For effective programmability, the system must be constrained to operate sequentially to ensure that unanticipated conflicts do not occur (for a discussion of problems in this area, see [18]). This means that most of the processors remain dormant at any given time. Of course, if a program is naturally decomposable into a number of independent parts, each having approximately the same running time, the dormancy can be greatly reduced. The running time can only be predicted in special cases, however, since predicting running times would be tantamount to determining whether programs are defined and thus to solving the halting problem.

The component level. For structural programmability, the number of types of switching primitives used must be finite, and each must have predefined switching capabilities. Consequently, these primitives cannot in general be adapted to the task at hand. As the number of types of components (primitives) increases, the number of components required to implement any given program typically decreases, since the functions per-

formed by the added primitives would not need to be simulated by combinations of already existing primitives. When our objective is to perform a highly selective information-processing task with the smallest number of components, an arbitrarily large repertoire of component types is obviously desirable. This is essentially the situation when the components are protein enzymes. There are at least 20^{300} different amino-acid sequences that could be fabricated by biological systems or by recombinant DNA techniques. Since the functional properties of each of these sequences arise through an energy-dependent folding process, however, they cannot be prescribed by a manageably small programmer's manual.

This inefficiency at the component level contributes to inefficiency at the network level. If components could always evolve to specifically suit the task at hand, networks could learn to use their resources in parallel. If we are willing to do without effective programmability in order to use components in parallel, the only reason for working with a restricted set of component types is the practical one that fabricating new primitives out of silicon is expensive. This limit of current technology is hardly relevant to biological systems, however, which can always use their genetic apparatus to generate and mass-produce new carbon polymers.[2]

The physical level. Structural programmability also cuts into the computing potential allowed by the principles of physics [2]. Recall that, according to Turing–Church, a structurally programmable physical realization of a formal computation process should be capable of simulating any physical process when space and time are unlimited. Space and time resources are never unlimited in the real world, however, and are in fact generally limited in different ways for the simulating system (the machine) and the simulated system (the physical process). A simulating system consisting of a set of particles constrained to execute formal computations could never keep pace with a simulated system consisting of the same number of particles; if the simulating system is structurally programmable, it is even further constrained.[3]

According to known force laws, n^2 interactions could conceivably contribute to the behavior of a system consisting of n particles. The potential parallelism here is at least n-fold greater than that obtainable by one n-processor machine of conventional design. For the system to exhibit formal computational behavior, constraints must be introduced to suppress a large fraction of the interactions (the importance of constraints, in particular nonholonomic constraints, is emphasized in [37]). These constraints must ensure that the time development of the system is not described by an analytic function (since the execution sequence of a formal computation must in general be independent of any subsequence). They must also ensure that the system's time development corresponds to a classically picturable execution sequence, that is, quantum mechanics must be irrelevant to the system as a whole. In fact, all dynamical processes involving the system as a whole must be suppressed, in contrast to most systems in nature. Since the components must be in one of a number of distinct states (corresponding, say, to 0 and 1), the number of distinct quantum states available is a potential limitation. Distinct quantum states contribute to the reliability of storage and switching, requirements that do not apply to all physical systems. To achieve structural programmability, engineers must impose additional enforcement constraints to ensure that only interactions that allow each component to realize its predefined function can occur and that prevent the occurrence of all other interactions. The limit of n-fold parallelism in today's machines is a consequence of these enforcement constraints.

These constraints are a contributing factor to physical limitations on present-day computers. Communication among components is limited by signal velocity and ultimately by the velocity of light. Signal velocity is already becoming a limiting factor in some supercomputers. Structural programmability generally requires special functions that could otherwise be performed by a single specifically adapted component to be duplicated by a combination of components, thus increasing the distance over which signals must be sent. Brownian motion is also a physical limitation (on reliability), but only to the extent that we desire computation to fulfill preexisting specifications and if randomness is *not* one of these specifications. If randomness is desired, Brownian motion can actually be viewed as extremely heavy computation. For example, an enzyme exploring a substrate utilizes Brownian motion for tactile pattern recognition. Exploration of this kind would be incompatible with the enforcement constraints required for structural programmability.

The most important physical limitation on today's computers is heat export. Like living systems, computers feed on high-grade energy and give off heat. Transistors in present-day computers dissipate about 10^{10} kT per step. Enzymatic reactions in biological systems typically dissipate 10–100 kT per step. This is an impressive difference, especially when we consider that enzymes perform a tactile pattern-recognition task that would involve many steps for a digital computer. Careful examination reveals an even more remarkable fact about enzymatic computing: An enzyme actually performs its pattern-recognition task reversibly. At equilibrium each enzyme continues to recognize substrates and to make or break covalent bonds. Energy dissipation drives the reaction being catalyzed, rather than the pattern-recognition activity of the enzyme. Driving the reaction forward is essential for coupling the pattern-

[2] The limitation on the number of primitives is not contradicted by microprogrammable systems. These are properly thought of as general-purpose machines comprising general-purpose components, not components specifically trimmed for a particular task.

[3] This physical view of computing has the paradoxical implication that the ultimate physical capabilities of nature must forever escape a constructive mathematical treatment.

recognition activity of the enzyme to the control of macroscopic processes, but not for the pattern-recognition activity itself.

In recent years Bennett, Fredkin, and Landauer [1, 30] have produced a surprising result that connects in an interesting way to this fact about enzymes. This result is that physical realizations of formal computation processes can in principle proceed with arbitrarily low dissipation when speed, reliability, and required memory space are not considerations. The construction they used to demonstrate this is based on a thermodynamically reversible logic gate. It is also possible to view this result in the light of the Turing–Church thesis: A logically reversible Turing process that is computation universal is in principle possible. It is only necessary to record all the intermediate states on the memory tape, record the answer, and then use the memory tape to recover all the initial information. If a thermodynamically reversible realization of this logically reversible Turing machine were demonstrably impossible even in the limit of idealization, it would be impossible to use a digital process to simulate reversible processes in nature without losing reversibility. If it were impossible to duplicate a reversible process digitally without embedding it in an irreversible process, then a weakening of the strong interpretation of the Turing–Church thesis would be necessary [7].

The Bennett–Fredkin–Landauer result shows that there is no machine-independent thermodynamic limit to computing. Thermodynamic costs can be traded for other costs that can be reckoned in terms of components, reliability, and speed. If a machine is structurally programmable, these other costs are high, and hence the cost of adding more components or accepting less speed and reliability in order to reduce the dissipation would soon outweigh the advantage to be gained. If the system foregoes structural programmability, the balance changes, since the amount that must be invested to obtain a given decrease in dissipation is less. This is a probable explanation for the relatively low energy dissipation found in biological computing.

EFFICIENCY VERSUS COMPLEXITY

To properly appreciate the significance of this extra efficiency, it is useful to distinguish between polynomial-time and exponential-time problems [19]. A task falls into the polynomial-time class if the number of resources required to solve it increases as a low-order polynomial function of problem size (say as n^2, where n is any reasonable measure of problem size). A problem is in the exponential-time class if the resources required to solve it increase at least exponentially with problem size, say as 2^n. Problems involving this type of increase entail a *combinatorial explosion.*

In the explosive 2^n case, a 10^{10}-fold increase in resources allows an additive increase in problem size of at most 33, a very modest return for a major investment. In the polynomial-time n^2 case, the same 10^{10}-fold increase in resources allows the problem size to increase by as much as a factor of 10^5—a dramatically significant improvement. It is clear, therefore, that boosting computational resources will not make an indent on the combinatorial explosion, but that it can be very helpful for problems admitting an efficient solution. There is an important qualifier, however. In general, the increase in problem size is much less than 10^5. The extent to which this limit can be approached depends on the efficiency with which computing resources can be coupled to a problem's solution. In structurally programmable computers, the efficiency of coupling is low, and an increase in computing power along conventional lines is not necessarily as valuable as we might at first assume. If most machine resources are dormant at any given time, an enormous increase in parallelism will not be worth the cost. Systems that opt for efficiency and evolutionary adaptability would be better suited to coupling increases in computational resources to increases in problem size.

Could analog processes or other dynamical processes counteract the combinatorial explosion? It is not possible to answer this question a priori, but it is possible to focus the issue more precisely. Suppose a particular dynamical process could be used for solving a large exponential-time problem. Furthermore, suppose that this process could be simulated on a digital computer over a short interval of time. A constant factor speedup of the digital computer would then allow what is by hypothesis an exponential-time problem to be solved in polynomial time [15]. Thus if we assume that particular dynamical processes can be harnessed for solving exponential-time problems, we must also assume that these processes are refractory to digital simulation for even the smallest time slices.

The existence of dynamically exotic processes capable of solving exponential-time problems is of course a logical possibility. The phenomenon of turbulence would conceivably fall into this category. The Turing–Church thesis would not be contradicted by such processes since it does not matter, in terms of the thesis, whether the number of resources used for the digital simulation grows exponentially. The potential value of structurally nonprogrammable computers does not depend on this logical possibility, however. The value is actually greatest for problems admitting efficient solutions, for here the size of problems that are manageable grows the most. Indeed it may be that the dramatically different capabilities of biological organisms and von Neumann machines are largely due to the fact that the former are capable of solving much larger problems in the polynomial-time class. From the standpoint of constructing artificial computers out of new materials, the implication is clear: What is needed is not von Neumann computers fabricated from alternative materials, but new computer designs to address computing needs not efficiently met by von Neumann machines.[4]

[4] Of course, there may be other reasons for building von Neumann machines out of alternative materials, such as the desirability of machines resistant to pulses of electromagnetic radiation.

THE PRINCIPLE OF DOUBLE DYNAMICS

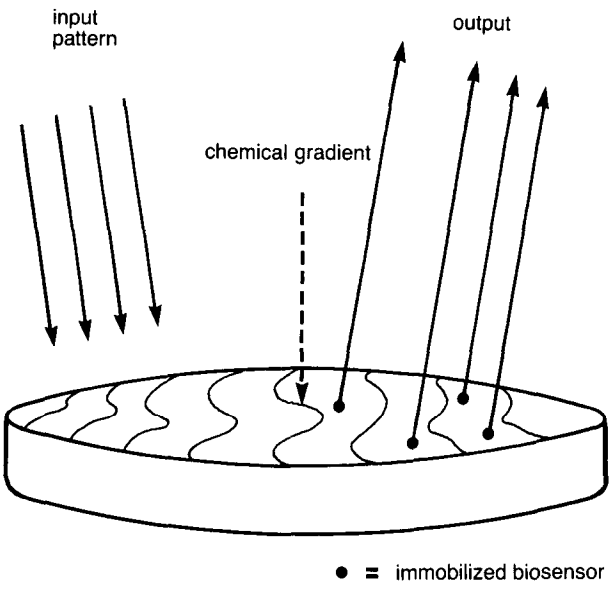
How could a structurally nonprogrammable system be organized to perform useful functions? Undoubtedly many guiding principles could be used for design in this area. However, I will focus on one, the *principle of double dynamics*, which seems to be particularly promising and which appears to play a role in biological information processing.

Double dynamics would exploit the trade-off principle and would be chemically implementable. As in conventional computing, we would need building-block modules to transform patterns of input signals into patterns of output signals. Since conventional computers operate on the programmability side of the trade-off principle, their building blocks perform this transformation in a manner completely describable in terms of a finite user's manual. In order to shift to the efficiency-adaptability side of the trade-off relation, transformation should combine elements of *continuity* and *gradual modifiability* with *decision making*.

Chemistry offers the tactile pattern-matching capability inherent in the conformation of proteins and other carbon polymers. Such polymers can be gradually modified and, with enzymes at least, supply the element of decision making. However, enzymes respond to surface properties such as the shapes of particular substrate molecules in their immediate environment. This tactile form of pattern recognition is unsuitable for the electrical, optical, and mechanical signals that are most conveniently presented to a module.

Chemistry also offers various kinds of dynamics involving reaction and diffusion that could transduce nontactile signals into a form that enzymes can recognize. These dynamics supply the continuity required for generalization, as well as an element of gradual transformability if the dynamics are structurally stable. The studies of Prigogine and his school [36] have shown that, in the presence of suitable nonlinearities, complex spatial and temporal patterns (known as dissipative structures) can be created in a chemical medium through the production of entropy, thereby providing a dynamic structure-formation principle that complements the energy-based self-assembly principle utilized by polymers possessing conformation. The advantage of dissipative dynamics is that it can process patterns of signals into surface structures in the form of chemical gradients.

The device pictured in Figure 7 combines these two powerful forms of pattern processing—enzymatic action and dissipative structures—to implement the principle of double dynamics. The enzymes are immobilized on a surface covered by, say, a photochemically sensitive medium. The nonlinear, dissipative dynamics of this medium serves to transduce spatial and temporal patterns of input signals (e.g., photons) into a chemical concentration gradient. This is the first level of dynamics. The chemical concentrations are *interpreted* by the enzymes, each of which can be thought of as a biosensor coupled to an output device. For concreteness we



The input pattern is a nontactile pattern such as of optical or electrical signals. This pattern is transduced into a chemical-concentration pattern that can be read out by immobilized biosensors, that is, by enzymes linked to an output device. The output either controls a process or is passed to a von Neumann computer. The chemical medium provides the upper level of dynamics. The readout enzymes provide the lower level. The time required for the device to transduce an input pattern into an output pattern could be much reduced if diffusion in a chemical medium were replaced by a mechanical propagation of signals in a network of structural proteins.

FIGURE 7. A Double-Dynamics Device

suppose that the action of the enzymes on their substrates in the reaction-diffusion layer is picked up spectroscopically. The action of the enzyme is thus a distinct, second level of dynamics that controls the output of the device. If it is sufficiently rich, the top level of reaction-diffusion dynamics converts different input-signal patterns into different chemical patterns. The device is "programmed" through variation at the enzymatic level of dynamics and selection on the basis of the pattern-processing performance of the whole system. The variations may involve the spatial arrangement of the enzymes or modifications in their response to a given chemical milieu. As long as the top level of dynamics is not initial-condition sensitive, similar input-signal patterns should often give rise to similar chemical patterns, thus conferring some power of generalization on the system.

The first potential use of such a device is as a preprocessor—a kind of eye—for a conventional von Neumann machine. The purpose would not be simply to transduce environmental stimuli, but to use the rich dynamics of the diffusion layer and the intelligence of the enzyme to code environmental patterns into a form

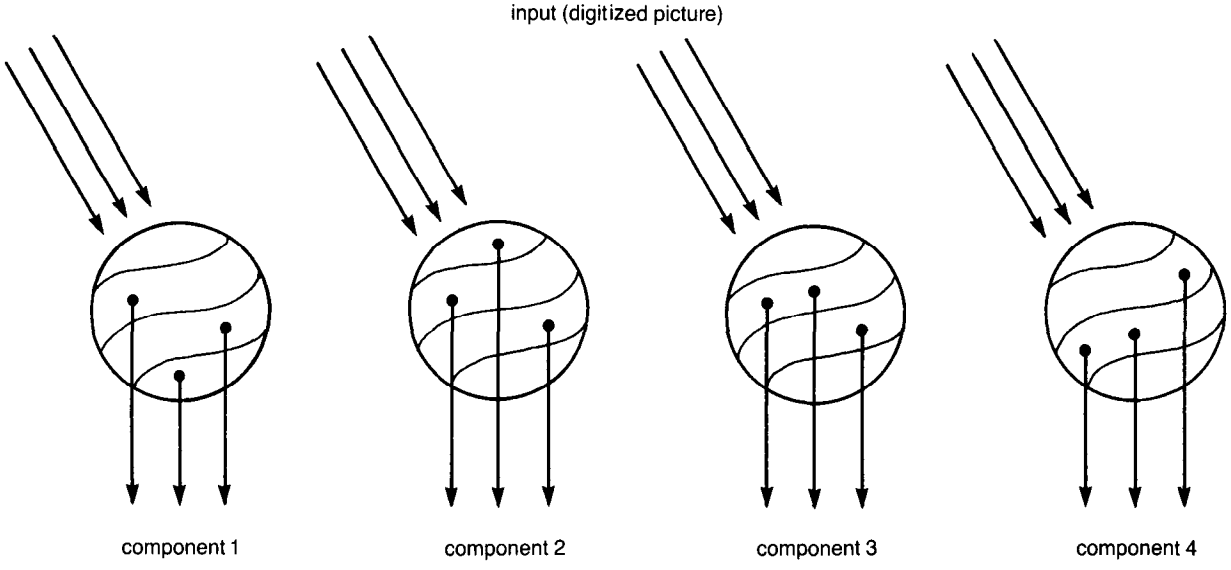
simple enough to be managed by a structurally programmable machine.

As a second step, we could directly couple the output to an effector organ—for instance, a robot arm. Decisions about the type of motion the arm makes are controlled through the von Neumann machine, but fine motion, in response to the particularities of the environmental situation, is controlled by the way the enzymes interpret this situation as it is represented in the dissipative dynamics of the reaction-diffusion layer. Figure 8 illustrates an expanded version of such an adaptive pattern-recognition and process-control system. The figure shows a bank of double-dynamics modules, each individually controlling a different part (or degree of freedom) of a robot arm. Each module receives the same environmental input and represents this input in the same dissipative pattern. The enzymes in each module are selected differently, though, so the timing and intensity of the outputs are different. Each module can be thought of as an analog of the motor element it controls. This shifts the problem of coordinating the whole arm (in the context of a particular type of action) to the problem of evolving suitable analog modules.

To ascertain the class of functions that a network of such enzymatically driven modules is in principle capable of computing, we can suppose, for simplification, that a threshold property is introduced to reduce the possible outputs to one or zero. This can be done in a variety of ways, but for present purposes it is sufficient to divide the module up into small regions and to ignore the enzymatic signal in any region if it is either above or below a prescribed strength. The resulting module is much less powerful than it might be, but is still more powerful than classical formal neurons of the McCulloch–Pitts type. These fire whenever the average of their inputs exceeds a threshold, whereas our dynamic modules would fire in response to particular input patterns (assuming that the reaction-diffusion dynamics are sufficiently rich and the enzymes sufficiently sensitive). However, we know that any finite automaton can be simulated by networks of McCulloch–Pitts neurons, even neurons as simple as the threshold-two elements in Figure 5. It follows that networks of more powerful enzymatic modules must be capable of simulating any finite automaton. The simulation is potentially more efficient since it does not have to work around predefined modules. Since each module is potentially unique, however, there can be no predefined construction manual.

This leads us to conclude that it is possible to build a computer that is computation universal but not structurally programmable (for the formal construction, see [9]). Since networks of enzymatically driven modules can simulate any finite automaton, it must be possible to use them to simulate an interpreter. Thus it would be possible to build a structurally nonprogrammable computer that is nevertheless effectively programmable at an interpretive level. There is a model for such a machine: People can read and follow rules despite the fact that the human brain, as a product of evolution, must be structurally nonprogrammable.

Turing machines compute a much larger class of functions than finite automatons. However, Turing machines with finite memory tapes—the realistic assumption—can always be simulated by finite automatons.



Each module receives the same input, but interprets it differently to control that component of the robot arm for which it is an analog. The modules are versions of the double-dynamics device shown in Figure 7.

FIGURE 8. An Array of Double-Dynamics Devices

Practically speaking, some sort of manipulable memory store is a great advantage. Modern digital computers use an addressable memory, that is, a memory in which each switch is located on a grid in a manner that allows it to be accessed individually. Alternative structures are possible in which patterns of activity of highly interconnected memory units play a role; this is probably what happens in the nervous system. Memory storage would be greatly increased by such multiple utilization of components, although powerful pattern-processing capabilities would be needed to manipulate such a distributed memory structure [10]. This is just the kind of capability that could be provided by enzyme-driven networks.

DOUBLE DYNAMICS IN NATURE

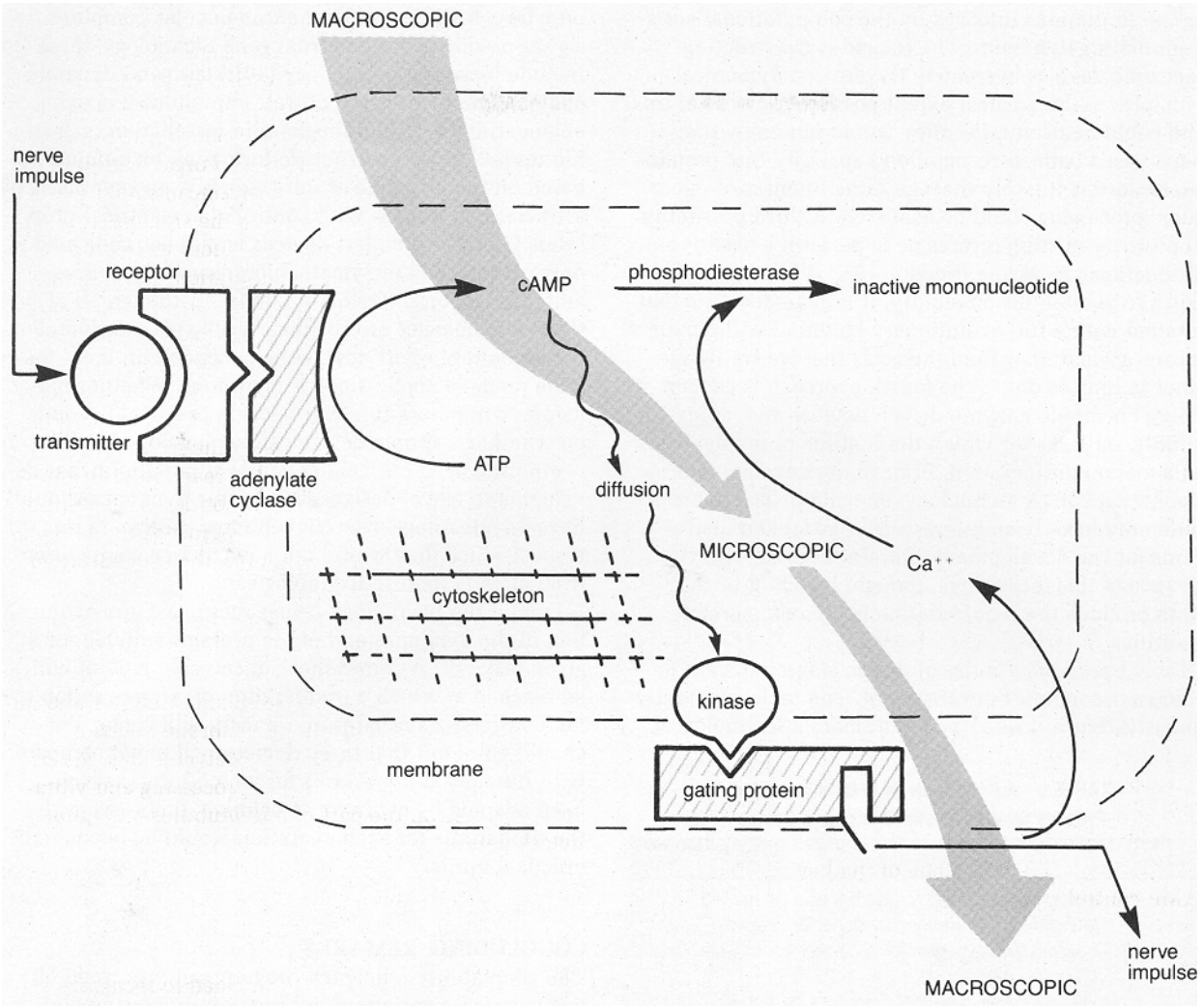
We have not been specific about the chemistry of man-made double-dynamics devices since none has as yet been produced. Our group at Wayne State University has conducted extensive simulation studies of devices that utilize the double-dynamics principle, however, and has developed evolutionary learning algorithms for communicating desired performance to them [25–27]. The reactions used in these simulations occur in real neurons and can therefore be specified with some degree of confidence.

The reactions involve molecules called second messengers, which serve as intracellular signal carriers. Second messengers such as cAMP, calcium, and possibly cGMP are now known to control nerve-impulse activity in some neurons. Some of the key reactions of the cAMP control system are illustrated in Figure 9 [20]. Presynaptic inputs activate receptors on the cell membrane that produce cAMP, which in turn acts on protein kinases. These kinases may activate gating proteins, which control the electrical activity of the membrane, or may act on other target proteins inside the neuron. If the neuron fires, calcium enters to activate the enzyme phosphodiesterase, which in turn resets the neuron to its initial state by converting cAMP to the inactive mononucleotide. There are a number of other proteins in this system that also play a controlling role, such as binding proteins that control messenger diffusion rates and an enzyme (phosphoprotein phosphatase) that reverses the activation of protein kinase. It is possible to experiment on this system by microinjecting cAMP into neurons and studying the effects on nerve-impulse activity. Liberman et al. [31, 32, 34] have reported an extensive series of such experiments, which indicate that cAMP injected into large neurons of a snail's central nervous system has a depolarizing effect, that is, increases the rate of neural firing. Other investigators have also reported effects, in some cases involving increased neural activity [28] and in others decreased activity (for a recent review, see [17]). The kinds of control that second messengers exert on nerve-impulse activity have not been definitively resolved, but the evidence that these molecules provide a two-way link between the nerve impulses and molecular processes inside neurons is consistently strong.

This two-way link also operates in the much simpler double-dynamics device illustrated in Figure 7. In both cases it serves to transduce macroscopic signals (e.g., nerve impulses or light rays) to a microscopic form at the molecular level of organization and then back to a macroscopic form. A major contribution to the computing occurs at the molecular level of organization in the device, where enzymes read the concentration gradients. The great advantage from the standpoint of efficiency is that this enormously reduces the heat production that accompanies the computation. If the same amount of computation were performed at the macroscopic level of nerve impulses, dissipation would become a significant limiting factor. This thermodynamic consideration suggests that a major contribution to biological computing occurs at the molecular level in the neuron, just as it does in the hypothetical device.

The obvious analogy is that the reaction-diffusion dynamics at the level of cyclic nucleotides is read out by kinases attached to gating proteins, just as the immobilized enzymes in Figure 7 read out the concentration gradients in the device. The rich internal structure of the neuron also suggests other more powerful possibilities. Recent microinjection experiments indicate that the effect of cAMP may in some instances be mediated by the cytoskeleton, a network of microtubules and microfilaments that support the membrane and run through the cell [23, 33]. It is conceivable that the potential for highly parallel signal processing and vibratory behavior on the part of microtubules and other cytoskeletal elements could play a significant role. In this case second messengers would most plausibly serve to initialize or modulate these cytoskeletal dynamics in a manner that reflects the pattern of input to the neuron. The interpretive function could be provided by a repertoire of readout molecules tuned to recognize significant patterns of cytoskeletal activity and capable of initiating processes (such as the production of cAMP or cGMP) to modulate the firing of the neuron. Although this is only speculation, it does serve to show that an enormous amount of computation could be allocated to the most microscopic molecular levels of organization in the double-dynamics framework.

The molecular mechanisms of gene action and heredity constitute a well-understood biological information-processing system whose organization is in some ways parallel to that of a double-dynamics device. The DNA sequence of genes is a structural code controlling an organism's phenotypic features under the interpretation of molecular decoding machinery (e.g., m-RNA, t-RNA). That the phenotypic features are often suited to the environment is due to the Darwinian mechanism of evolution. By comparison, the upper level dynamics of a double-dynamics device (or of a neuron) can be thought of as a dynamic code controlling the actions of the device (or organism) under the interpretation of molecular readout machinery (e.g., enzymes and molecular resonators). These actions are often suited to the environment as a consequence of evolution through variation and selection, though it is largely the readout enzymes that undergo evolutionary changes rather than the dynamic code. Readin mechanisms allow reg-



The firing of a presynaptic neuron releases a transmitter that combines with a receptor on the postsynaptic membrane. The receptor activates the enzyme adenylate cyclase, which catalyzes the production of cAMP from ATP. cAMP diffuses to another location on the membrane, where it activates a protein kinase, which in turn activates proteins, called gating proteins, that control the flow of ions across the membrane. If these ion channel proteins are sufficiently activated, the neuron fires, causing calcium (Ca^{++}) to be released, and activates the enzyme phosphodiesterase. Activated phosphodiesterase causes cAMP to be broken down to the inactive mononucleotide, effectively resetting the neuron to its initial state. The neuron continues to fire, however, until the kinase is deactivated by the enzyme phosphoprotein phosphatase (not shown). Many types of effector proteins, such as proteins associated with the cytoskeleton, can be linked to kinases and activated by cAMP. The time required for the neuron to transduce the spatiotemporal pattern of pulses impinging on it to a frequency-coded output would be much reduced if cAMP acted by initiating mechanical signals in the cytoskeleton. The important feature to note in this scheme is the flow of information from macroscopic to increasingly microscopic levels (sensory input → nerve impulses → chemical reactions → molecular events) and then back to macroscopic levels.

FIGURE 9. Key Reactions of the cAMP System of Intraneuronal Control

ulatory substances to alter the expression of DNA in response to the environment, just as second messengers may effect changes in cytoskeletal dynamics.

THE IMPETUS FROM BIOTECHNOLOGY

Enzyme-driven computing in biological systems depends on intricate processes of self-organization and self-maintenance. The complex sequence of reactions in the cyclic nucleotide system, not to mention the membrane and cytoskeletal architecture in which it is embedded, requires for support all the metabolic and self-regulatory processes of an entire organism. Creating such a self-enclosed support system artificially is clearly infeasible. How then could we ever exploit the efficiency-adaptability side of the trade-off relation?

There are at least four conceivable approaches. The first is working with virtual machines built on top of the von Neumann computer while consciously accept-

ing the limitations imposed by the computational costs of simulating dynamics. The second is constructing electronic devices to embody the desired dynamical principles to the greatest extent possible. Devices of this kind could be useful, but they would not enjoy the shape-based pattern-recognition capability that proteins possess, nor is it likely that the same fineness of microscopic processing could be achieved. A third possibility is culturing existing organisms to perform a useful information-processing function [22]. While it may be useful to pursue this possibility, it is generally true that a system with a full evolutionary potential will pursue its own goals, rather than the goals that we would attempt to impose on it. The fourth approach is concentrating on simple enzyme-driven devices and, at least initially, on tasks for which the lifetime of the device is not a major consideration. Prior to the development of recombinant DNA technology, enzyme-driven devices could only have been interesting as gedanken instruments for the development of brain models. With the advance of this technology, though, it should be feasible to produce the required proteins in commercial quantities.

Table I presents a series of design stages relevant to molecular-computer development. The table is patently futuristic, but it does serve to enumerate technologies

TABLE I. A Progressively Futuristic Design Process for a Molecular-Computing Device

Stage	Development
1	Develop a basic module like the one illustrated in Figure 7 (a simple transducer or preprocessing sense organ for a conventional digital computer).
2	Replace simple chemical gradients by richer dynamics (e.g., symmetry-breaking dissipative dynamic structures).
3	Perfect and mass-produce suitable biosensors using evolutionary methods (recombinant DNA techniques).
4	Embed a microskeleton in the module (e.g., using tessellations of immunoglobin molecules).
5	Develop biosensors that can use their specificity to self-assemble on the microskeleton.
6	Envelop the module in an artificial membrane.
7	Use the biosensors to control the electrical properties of the membrane.
8	Use the tessellation to transmit and integrate mechanical signals in a parallel manner. Use proteins as structural effectors to control the dynamics of the tessellation.
9	Develop (by evolutionary procedures) a large library of special-purpose modules.
10	Network these high-level primitives into a rich variety of special-purpose modules.
11	Use the pattern-processing properties of these modules to facilitate the manipulation of highly distributed memory systems.
12	Enclose the evolutionary adaptation process into the system itself.

on whose further development molecular-computer design is predicated. Aside from gene technology, these include biosensor technology [46], static and dynamic chemical dissipative structures, immobilized enzyme engineering [47], immunoglobulin tessellation techniques [42] or other structure-formation techniques based on the principle of self-assembly, and synthesis of artificial membranes with controllable electrical properties [39]. The simplest devices would be predicated only on gene and enzymatic biosensor technologies, and on reaction-diffusion dynamics. Although all of these technologies are in an early stage of development, the amount of effort now being expended on them for a wide range of applications is enormous. Whether molecular computers can ever become practical depends on whether a simple device can be developed to fill a computing need critical enough to stimulate further development. Novel designs like double dynamics should have an advantage over conventional designs in this respect, since they would not have to face competition from already mature technologies.

Even if the technology comes into place, the evolution of the dynamics and of the proteins suitable for a given task will require time. Conceivably a point will be reached at which a proliferation of devices suitable for a variety of special-purpose tasks will occur. We should point out that these devices will never compete with humans at those tasks for which humans have been adapted by evolution. Technical problems aside, the time frame for such evolution would be beyond all practical limits.

CONCLUDING REMARKS

The adaptability-efficiency-programmability trade-off ties together a variety of information-processing phenomena exhibited by both technical and biological systems. The technical problems that must be addressed in order to exploit this trade-off with molecular and chemical mechanisms have an admittedly forbidding aspect. However, it is noteworthy that the required biological technologies, especially the ability to produce specific proteins on a massive scale, are likely to develop and in time to significantly influence the chemical industry, agriculture, and medicine. While it is hard to imagine that the development of a commercially viable molecular computer is imminent, it is equally difficult to imagine a future world in which computing remains aloof from the prevailing technical infrastructure.

There are two respects in which the trade-off principle is of immediate interest [4]. First, gene technology is expanding the possibilities for biological design. If this technology is used as a means for prescriptive planning of biological systems similar to the prescriptive control exerted over a von Neumann computer, the trade-off principle predicts that we will inevitably lose evolutionary adaptability.

The second area of immediate concern is the use of the von Neumann computer or of any other structur-

AN EMERGING CONSENSUS ON MOLECULAR COMPUTING?

Michael Conrad proposes that the information-processing capabilities of organic molecules could be used in computers in place of digital switching primitives. So far, though, there is no clear consensus as to the viability of this concept or the best strategy for molecular computing. There have now been two international workshops on fabricating both inorganic and biologically derived molecular circuitlike devices, as well as a conference on chemically based computer design; Conrad's paper is in fact based on his keynote address from that conference.

According to Kevin Ulmer, vice-president for advanced technology at Genex Corporation, a major genetic engineering firm, however, "no one could legitimately claim to be working on a biologically derived molecular electronic device right now. In fact, there is hardly any experimental work of consequence being conducted even for nonbiological approaches."

The development of molecular-computing devices is currently limited both by materials science and analytical methods. As to the specific problem of adapting proteins, "it will be necessary to develop a protein engineering technology for altering the structure and function of proteins in a precise and predictable fashion before anything else can be done," says Ulmer.

Ulmer believes that a likely first step would be to design hybrid devices—solid-state biological sensors built from both conventional semiconductors and proteins. This would be much easier than working on the molecular processing level. "It's the bulk electronic properties of biological materials that we should be looking at. Building individual switches based on molecules is simply beyond our present capabilities."

Mark Stefik, a principal scientist in the Intelligent Systems Laboratory at Xerox PARC, doubts that proteins could make computers more adaptable: "Adaptability is an emergent property at much higher levels than enzymes in living systems. The interesting information processes that underlie evolution do not arise out of the properties of isolated molecules, but rather out of the structure and organization of living cells." Furthermore, the "architecture" of biological systems remains only partially understood. "There may be very special possibilities for miniaturization in molecular technology. But we're a long way from understanding any viable principles of operation at this point."

Danny Hillis, a founding scientist at Thinking Machines Corporation in Cambridge, Massachusetts, feels that a technological bridge between computing and biology must be built before molecular devices can even be attempted: "If specific proteins—receptors or antibodies—could be built from scratch, then a whole range of applications could be opened up. In the meantime, though, there is no technological foundation for bringing organic materials directly into computers."

In lieu of this, however, there are a number of approaches that might successfully combine computing and biology. According to Ulmer, "If general solid-state biological sensors could be developed that were sufficiently small, sensitive, stable, and specific, they could be useful in a number of ways. We could use such biosensors as the 'noses' and 'taste buds' of computers. This would complement the work currently being done in robotics to give computers a crude form of vision, a sense of touch, and an ability to manipulate objects." Ulmer also suggests that the interesting electronic and optical properties of some protein crystals could be utilized for electro-optical devices.

David Waltz, a senior scientist at Thinking Machines Corporation, lists four possible advantages that molecular-computing devices might have if they can ever be developed:

- Current computer technologies are essentially planar, and are thus limited in overall density and use of three-dimensional space.
- There are limits to miniaturization with current technologies—wires cannot be made much smaller without becoming subject to destruction by stray cosmic radiation or semiconductor impurities. Molecular-computing devices, besides being much smaller than anything we can make with current technology, may even offer possibilities for self-repair or self-regeneration.
- Certain computations may, in fact, as Conrad suggests, be better suited to molecular devices.
- Molecular devices may force us to explore the relationships among programmability, adaptability, and efficiency, and so lead to new theoretical insights on computation and intelligence, and in the process to new computing engines that will be superior in certain ways to existing computers.

ally programmable computing system. According to the trade-off principle, these systems are inherently less adaptable than biological systems and inherently less efficient for the performance of a wide variety of critical computing tasks. It cannot be doubted that the controllable information-processing power of current computers is an enormously valuable new resource and that the current expansion of this form of computing can have an enhancing effect on various aspects of human activity. If the trade-off principle is correct, however, there are aspects of human intelligence that digital computers will never be powerful enough to duplicate. The trade-off principle does not contradict the claim of Turing and Church, but it clearly contradicts the expectations of many advocates of artificial intelligence. To productively incorporate computing into the infrastructure of society, it is necessary to understand the relationships among adaptability, efficiency, and programmability. Here the study of molecular-computer designs has immediate practical significance for computer science independent of whether or not such designs can be concretized either in the near or distant future.

Acknowledgments. This article is a written version of the keynote address delivered at the Conference on Chemically-Based Computer Design, sponsored by the National Science Foundation and the UCLA Crump Institute for Medical Engineering, and held in October 1983. I am indebted to F. E. Yates, the chief organizer, and to other participants for valuable comments that enabled me to improve the presentation. The conference proposal was based in part on the report "Biochips: A Feasibility Study," prepared by myself,

C. Friedlander, and K. Akingbehin, and supported by National Geno Sciences. Wayne State University faculty participating in the study group also included R. Kampfner and R. Rada (from the Computer Science Department) and R. Arking, H. Burr, V. Hari, D. Njus, A. Seigel, and A. Sodja (from the Biological Sciences Department). J. Taylor facilitated the study and contributed to it. I also acknowledge useful conversation with H. M. Hastings and support from the Computer Science Section of the National Science Foundation (Grant MCS-82-05423, *Evolutionary Programming Techniques for Adaptive Information Processing*).

REFERENCES

1. Bennett, C.H. Logical reversibility of computation. *IBM J. Res. Dev. 17*, 6 (Nov. 1973), 525–532.
2. Bremermann, H.J. Optimization through evolution and recombination. In *Self-Organizing Systems*, M.C. Yovits, G.T. Jacobi, and G.D. Goldstein, Eds. Spartan Books, Washington, D.C., 1962, pp. 93–106.
3. Carter, F.L., Ed. *Molecular Electronic Devices*. Marcel Dekker, New York, 1982.
4. Conrad, M. *Adaptability*. Plenum Publishing Corp., New York, 1983, pp. 234, 347–370.
5. Conrad, M. Evolution of the adaptive landscape. In *Theoretical Approaches to Complex Systems*, R. Heim and G. Palm, Eds. Springer-Verlag, New York, 1978.
6. Conrad, M. Information processing in molecular systems. *BioSystems 5*, 1 (Aug. 1972), 1–14.
7. Conrad, M. Microscopic-macroscopic interface in biological information processing. *BioSystems 16*, 3–4 (1984), 345–363.
8. Conrad, M. Molecular automata. In *Physics and Mathematics of the Nervous System*, M. Conrad, W. Güttinger, and M. Dal Cin, Eds. Springer-Verlag, New York, 1974, pp. 419–430.
9. Conrad, M. Molecular information processing in the central nervous system, part I. In *Physics and Mathematics of the Nervous System*, M. Conrad, W. Güttinger, and M. Dal Cin, Eds. Springer-Verlag, New York, 1974, pp. 82–107.
10. Conrad, M. Molecular information structures in the brain. *J. Neurosci. Res. 2*, 3 (1976), 233–254.
11. Conrad, M. Mutation-absorption model of the enzyme. *Bull. Math. Biol. 41*, 3 (1979), 387–405.
12. Conrad, M. Natural selection and the evolution of neutralism. *BioSystems 15*, 1 (1982), 83–85.
13. Conrad, M. The importance of molecular hierarchy in information processing. In *Towards a Theoretical Biology*, vol. 4, C.H. Waddington, Ed. Edinburgh University Press, United Kingdom, 1972, pp. 222–228.
14. Conrad, M. The limits of biological simulation. *J. Theor. Biol. 45*, 2 (June 1974), 585–590.
15. Conrad, M., and Rosenthal, A. Limits on the computing power of biological systems. *Bull. Math. Biol. 43*, 1 (1980), 59–67.
16. Davis, M. *Computability and Unsolvability*. McGraw-Hill, New York, 1958.
17. Drummond, G.I. Cyclic nucleotides in the nervous system. In *Advances in Cyclic Nucleotide Research*, vol. 15, P. Greengard and G.A. Robison, Eds. Raven Press, New York, 1983, pp. 373–494.
18. Enslow, P.H. Multiprocessor organization—A survey. *Comput. Surv. 9*, 1 (Mar. 1977), 103–129.
19. Garey, M.R., and Johnson, D.S. *Computers and Intractability*. W.H. Freeman and Co., San Francisco, Calif., 1979.
20. Greengard, P. *Cyclic Nucleotides, Phosphorylated Proteins, and Neuronal Function*. Raven Press, New York, 1978.
21. Gutmann, F., and Lyons, L.E. *Organic Semiconductors, Part A*. R.E. Krieger Publishing Co., Melbourne, Fla., 1981.
22. Haddon, R., and Lamola, A. The organic computer: Can microchips be built by bacteria? *The Sciences 23*, 3 (May–June 1983), 40–45.
23. Hameroff, S.R., and Watt, R.C. Information processing in microtubules. *J. Theor. Biol. 98*, 4 (Oct. 1982), 549–561.
24. Hofstadter, D. *Gödel, Escher, Bach: An Eternal Golden Braid*. Vintage Books, New York, 1980.
25. Kampfner, R., and Conrad, M. Computational modeling of evolutionary learning processes in the brain. *Bull. Math. Biol. 45*, 6 (1983), 931–968.
26. Kampfner, R., and Conrad, M. Sequential behavior and stability properties of enzymatic neuron networks. *Bull. Math. Biol. 45*, 6 (1983), 969–980.
27. Kirby, K.G., and Conrad, M. The enzymatic neuron as a reaction-diffusion network of cyclic nucleotides. *Bull. Math. Biol. 46*, 5–6 (1984), 765–783.
28. Kononenko, N.I., Kostyuk, P.G., and Sherbatko, A.D. The effect of intracellular cAMP injections on stationary membrane conductance and voltage- and time-dependent ionic currents in identified snail neurons. *Brain Res. 268*, 2 (June 1983), 321–338.
29. Kuck, D. *The Structure of Computers and Computations*, vol. 1. John Wiley & Sons, New York, 1978.
30. Landauer, R. Uncertainty principle and minimal energy dissipation in the computer. *Int. J. Theor. Phys. 21*, 3–4 (Apr. 1982), 283–297.
31. Liberman, E.A., Minina, S.V., and Golubtsov, K.V. The study of the metabolic synapse. I. Effect of intracellular microinjection of 3′,5′-AMP. *Biophysics 20*, 3 (1975), 457–463.
32. Liberman, E.A., Minina, S.V., and Shklovsky-Kordy, N.E. Neuron membrane depolarization by 3′,5′-adenosine monophosphate and its possible role in neuron molecular cell computer (mcc) action. *Biophysics 23*, 2 (1978), 308–314.
33. Liberman, E.A., Minina, S.V., Shklovsky-Kordy, N.E., and Conrad, M. Change of mechanical parameters as a possible means for information processing by the neuron. *Biophysics 27*, 5 (1982), 906–915.
34. Liberman, E.A., Minina, S.V., Shklovsky-Kordy, N.E., and Conrad, M. Microinjection of cyclic nucleotides provides evidence for a diffusional mechanism of intraneuronal control. *BioSystems 15*, 2 (1982), 127–132.
35. Minsky, M. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, N.J., 1967.
36. Nicolis, G., and Prigogine, I. *Self-Organization in Nonequilibrium Systems*. Wiley-Interscience, New York, 1977.
37. Pattee, H.H. Physical problems of decision-making constraints. In *The Physical Principles of Neuronal and Organismic Behavior*, M. Conrad and M. Magar, Eds. Gordon and Breach, Science Publishers, New York, 1973, pp. 217–225.
38. Poston, T., and Stewart, I. *Catastrophe Theory and its Applications*. Pitman Publishing, Marshfield, Mass., 1978.
39. Przybylski, A.T., Statten, W.P., Syren, R.M., and Fox, S.W. Membrane, action, and oscillatory potentials in simulated protocells. *Naturwissenschaften 69*, 12 (Dec. 1982), 561–563.
40. Samuel, A.L. Some studies of machine learning using the game of checkers. *IBM J. Res. Dev. 3*, 3 (July 1959), 211–229.
41. Street, G.B., and Clarke, T.C. Conducting polymers: A review of recent work. *IBM J. Res. Dev. 25*, 1 (Jan. 1981), 51–57.
42. Uzgiris, E.E., and Kornberg, R.D. Two-dimensional crystallization technique for imaging macromolecules, with application to antigen-antibody-complement complexes. *Nature 301*, 589613 (Jan. 1983), 125–129.
43. Volkenstein, M.V. *Physics and Biology*. Academic Press, New York, 1982.
44. Waddington, C.H. *The Strategy of Genes*. George Allen and Unwin, London, 1957.
45. Wills, C. Production of yeast alcohol dehydrogenase isoenzymes by selection. *Nature 261*, 5555 (May 1976), 26–29.
46. Wingard, L.B., Jr. Immobilized enzyme electrodes for glucose determination for the artificial pancreas. *Fed. Proc. 42*, 2 (Feb. 1983), 271–272.
47. Wingard, L.B., Jr., Berezin, I.V., and Klysov, A.A., Eds. *Enzyme Engineering*. Plenum Publishing Corp., New York, 1980.
48. Wittgenstein, L. *Philosophical Investigations*. MacMillan & Co., London, 1953.

CR Categories and Subject Descriptors: C.1.3 [**Processor Architectures**]: Other Architecture Styles—*adaptable architectures*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation; I.2.m [**Artificial Intelligence**]: Miscellaneous
General Terms: Design, Theory
Additional Key Words and Phrases: biological information processing, evolutionary programming, learning, modes of computation, molecular computers, pattern recognition, trade-offs among complexity measures

Author's Present Address: Michael Conrad, Dept. of Computer Science, Wayne State University, Detroit, MI 48202.