International Journal of Theoretical Physics, Vol. 21, No. 12, 1982

The Thermodynamics of Computation—a Review

Charles H. Bennett

IBM Watson Research Center, Yorktown Heights, New York 10598

Received May 8, 1981

Computers may be thought of as engines for transforming free energy into waste heat and mathematical work. Existing electronic computers dissipate energy vastly in excess of the mean thermal energy kT, for purposes such as maintaining volatile storage devices in a bistable condition, synchronizing and standardizing signals, and maximizing switching speed. On the other hand, recent models due to Fredkin and Toffoli show that in principle a computer could compute at finite speed with zero energy dissipation and zero error. In these models, a simple assemblage of simple but idealized mechanical parts (e.g., hard spheres and flat plates) determines a ballistic trajectory isomorphic with the desired computation, a trajectory therefore not foreseen in detail by the builder of the computer. In a classical or semiclassical setting, ballistic models are unrealistic because they require the parts to be assembled with perfect precision and isolated from thermal noise, which would eventually randomize the trajectory and lead to errors. Possibly quantum effects could be exploited to prevent this undesired equipartition of the kinetic energy. Another family of models may be called Brownian computers, because they allow thermal noise to influence the trajectory so strongly that it becomes a random walk through the entire accessible (low-potential-energy) portion of the computer's configuration space. In these computers, a simple assemblage of simple parts determines a low-energy labyrinth isomorphic to the desired computation, through which the system executes its random walk, with a slight drift velocity due to a weak driving force in the direction of forward computation. In return for their greater realism, Brownian models are more dissipative than ballistic ones: the drift velocity is proportional to the driving force, and hence the energy dissipated approaches zero only in the limit of zero speed. In this regard Brownian models resemble the traditional apparatus of thermodynamic thought experiments, where reversibility is also typically only attainable in the limit of zero speed. The enzymatic apparatus of DNA replication, transcription, and translation appear to be nature's closest approach to a Brownian computer, dissipating $20–100kT$ per step. Both the ballistic and Brownian computers require a change in programming style: computations must be rendered *logically* reversible, so that no machine state has more than one logical predecessor. In a ballistic computer, the merging of two trajectories clearly cannot be brought about by purely conservative forces; in a Brownian computer, any extensive amount of merging of computation paths

would cause the Brownian computer to spend most of its time bogged down in extraneous predecessors of states on the intended path, unless an extra driving force of $kT\ln 2$ were applied (and dissipated) at each merge point. The mathematical means of rendering a computation logically reversible (e.g., creation and annihilation of a history file) will be discussed. The old Maxwell's demon problem is discussed in the light of the relation between logical and thermodynamic reversibility: the essential irreversible step, which prevents the demon from breaking the second law, is not the making of a measurement (which in principle can be done reversibly) but rather the logically irreversible act of erasing the record of one measurement to make room for the next. Converse to the rule that logically irreversible operations on data require an entropy increase elsewhere in the computer is the fact that a tape full of zeros, or one containing some computable pseudorandom sequence such as pi, has fuel value and can be made to do useful thermodynamic work as it randomizes itself. A tape containing an algorithmically random sequence lacks this ability.

1. INTRODUCTION

The digital computer may be thought of as an engine that dissipates energy in order to perform mathematical work. Early workers naturally wondered whether there might be a fundamental thermodynamic limit to the efficiency of such engines, independent of hardware. Typical of early thinking in this area was the assertion by von Neumann, quoted from a 1949 lecture (von Neumann, 1966), that a computer operating at temperature T must dissipate at least $kT\ln 2$ (about 3×10^{-21} J at room temperature), "per elementary act of information, that is per elementary decision of a two-way alternative and per elementary transmittal of one unit of information." Brillouin (1962) came to a similar conclusion by analyzing a thought experiment involving detection of holes in a punched tape by photons, and argued further that the energy dissipation must increase with the reliability of the measurement, being approximately $kT\ln(1/\eta)$ for a measurement with error probability η. These conjectures have a certain plausibility, in view of the quantitative relation between entropy and information exemplified by Maxwell's demon (Szilard, 1929), and the fact that each classical degree of freedom used to store a bit of information, e.g., the charge in a capacitor, suffers from kT of thermal noise energy, which seemingly would have to be overcome in order to read or manipulate the bit reliably. However, it is now known that computers can in principle do an arbitrarily large amount of reliable computation per kT of energy dissipated. In retrospect, it is hardly surprising that computation, like a complex, multistep industrial process, can in principle be accomplished with arbitrarily little waste, i.e., at thermodynamic cost only marginally greater than the difference in thermodynamic potential (if any) between its input and output. The belief that computation has an irreducible entropy cost per

step may have been due to a failure to distinguish sufficiently between dissipation (an irreversible net increase in entropy) and reversible transfers of entropy.

Though they are several orders of magnitude more efficient than the first electronic computers, today's computers still dissipate vast amounts of energy compared to kT. Probably the most conspicuous waste occurs in volatile memory devices, such as TTL flip-flops, which dissipate energy continuously even when the information in them is not being used. Dissipative storage is a convenience rather than a necessity: magnetic cores, CMOS, and Josephson junctions exemplify devices that dissipate only or chiefly when they are being switched. A more basic reason for the inefficiency of existing computers is the macroscopic size and inertia of their components, which therefore require macroscopic amounts of energy to switch quickly. This energy (e.g., the energy in an electrical pulse sent from one component to another) could in principle be saved and reused, but in practice it is easier to dissipate it and form the next pulse from new energy, just as it is usually more practical to stop a moving vehicle with brakes than by saving its kinetic energy in a flywheel. Macroscopic size also explains the poor efficiency of neurons, which dissipate about $10^{11}kT$ per discharge. On the other hand, the molecular apparatus of DNA replication, transcription, and protein synthesis, whose components are truly microscopic, has a relatively high energy efficiency, dissipating $20–100kT$ per nucleotide or amino acid inserted under physiological conditions.

Several models of thermodynamically reversible computation have been proposed. The most spectacular are the ballistic models of Fredkin and Toffoli (1982), which can compute at finite speed with zero energy dissipation. Less spectacular but perhaps more physically realistic are the Brownian models developed by Bennett (1973; see also below) following earlier work of Landauer and Keyes (1970), which approach zero dissipation only in the limit of zero speed. Likharev (1982) describes a scheme for reversible Brownian computing using Josephson devices.

Mathematically, the notion of a computer is well characterized. A large class of reasonable models of serial or parallel step-by-step computation, including Turing machines, random access machines, cellular automata, and tree automata, has been shown to be capable of simulating one another and therefore to define the same class of computable functions. In order to permit arbitrarily large computations, certain parts of these models (e.g., memory) are allowed to be infinite or indefinitely extendable, but the machine state must remain finitely describable throughout the computation. This requirement excludes "computers" whose memories contain prestored answers to infinitely many questions. An analogous requirement for a strictly finite computer, e.g, a logic net constructed of finitely many gates, would be that it be able to perform computations more complex than those

that went into designing it. Models that are reasonable in the further sense of not allowing exponentially growing parallelism (e.g., in a d-dimensional cellular automaton, the effective degree of parallelism is bounded by the dth power of time) can generally simulate one another in polynomial time and linear space (in the jargon of computational complexity, time means number of machine cycles and space means number of bits of memory used). For this class of models, not only the computability of functions, but their rough level of difficulty (e.g., polynomial vs. exponential time in the size of the argument) are therefore model independent. Figure 1 reviews the Turing machine model of computation, on which several physical models of Section 3 will be based.

For time development of a physical system to be used for digital computation, there must be a reasonable mapping between the discrete logical states of the computation and the generally continuous mechanical states of the apparatus. In particular, as Toffoli suggests (1981), distinct logical variables describing the computer's mathematical state (i.e., the contents of a bit of memory, the location of a Turing machine's read/write head) ought to be embedded in distinct dynamical variables of the computer's physical state.

2. BALLISTIC COMPUTERS

The recent "ballistic" computation model of Fredkin and Toffoli (1982), shows that, in principle, a somewhat idealized apparatus can compute without dissipating the kinetic energy of its signals. In this model, a simple assemblage of simple but idealized mechanical parts (hard spheres colliding with each other and with fixed reflective barriers) determines a ballistic trajectory isomorphic with the desired computation. In more detail (Figure 2), the input end of a ballistic computer consists of a "starting line," like the starting line of a horse race, across which a number of hard spheres ("balls") are simultaneously fired straight forward into the computer with precisely equal velocity. There is a ball at each position in the starting line corresponding to a binary 1 in the input; at each position corresponding to a 0, no ball is fired. The computer itself has no moving parts, but contains a number of fixed barriers ("mirrors") with which the balls collide and which cause the balls to collide with each other. The collisions are elastic, and between collisions the balls travel in straight lines with constant velocity, in accord with Newton's second law. After a certain time, all the balls simultaneously emerge across a "finish line" similar to the starting line, with the presence or absence of balls again signifying the digits of the output. Within the computer, the mirrors perform the role of the logic gates of a conventional electronic computer, with the balls serving as signals.

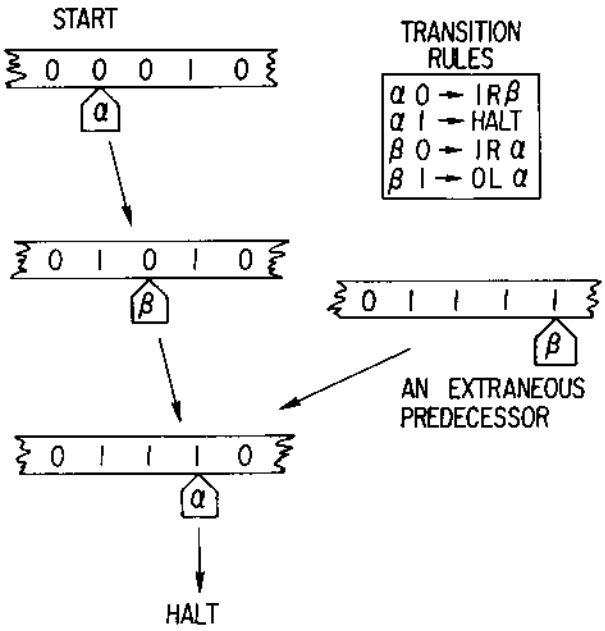

Fig. 1. An elementary mathematical model of computation, the Turing machine, consists of an infinite tape scanned by a movable finite automaton or "head," that can read or write one bit at a time, and shift a unit distance left or right along the tape. In order to remember what it is doing from one machine cycle to the next, the Turing machine head has a finite number of distinct internal states (here two: α and β). The Turing machine's behavior is governed by a fixed set of transition rules that indicate, for each combination of head state and scanned tape symbol, the new tape symbol to be written, the shift direction, and a new head state. The figure shows a short computation in which the machine has converted the input 00010, originally furnished on its tape, into the output 01110, and then halted. This Turing machine, because of its limited number of head states, can do only trivial computations; however, slightly more complicated machines, with a few dozen head states and correspondingly more transition rules, are "universal," i.e., capable of simulating any computer, even one much larger and more complicated than themselves. They do this by using the unlimited tape to store a coded representation of the larger machine's complete logical state, and breaking down each of the larger machine's machine cycles into many small steps, each simple enough to be performed by the Turing machine head. The configuration labeled "extraneous predecessor" is not part of the computation, but illustrates the fact that typical Turing machines, like other computers, often throw away information about their past, by making a transition into a logical state whose predecessor is ambiguous. This so-called "logical irreversibility" has an important bearing on the thermodynamics of computation, discussed in Section 4.

It is clear that such an apparatus cannot implement all Boolean functions: only functions that are conservative (the number of ones in the output equals the number of ones in the input) and bijective (to each output there corresponds one and only one input) can be implemented; but as Fredkin and Toffoli (1982) and Toffoli (1981) show, an arbitrarily Boolean function can be embedded in a conservative, bijective function without too much trouble.

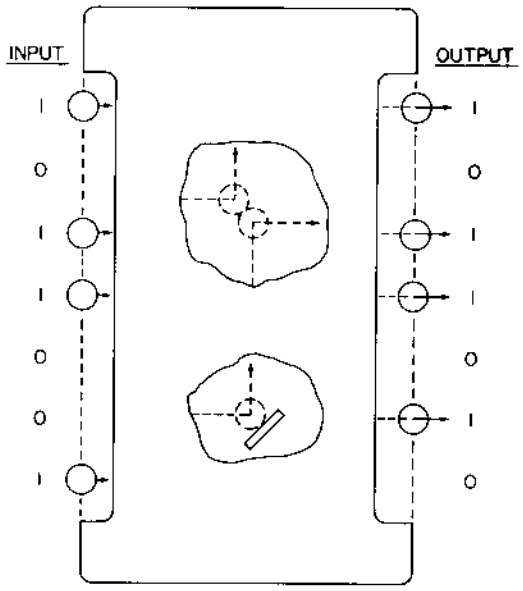

Fig. 2. Ballistic computer of Fredkin and Toffoli. In this example, the arrangement of mirrors inside the box is such that, when any five-bit number (here 13) is presented in the first five input positions, followed by 01 in the last two, the same five-bit number will appear in the first five output positions, followed by 01 if the number is composite, or 10 if the number is prime. The inclusion of the input as part of the output, and the use of two unlike bits to encode the desired answer, illustrate the embedding of an irreversible Boolean function into one that is reversible and conservative.

The two chief drawbacks of the ballistic computer are the sensitivity of its trajectory to small perturbations, and difficulty of making the collisions truly elastic. Because the balls are convex, small errors in their initial positions and velocities, or errors introduced later (e.g., by imperfect alignment of the mirrors) are amplified by roughly a factor of 2 at each collision between balls. Thus an initial random error of one part in 10^{15} in position and velocity, roughly what one would expect for billiard balls on the basis of the uncertainty principle, would cause the trajectory to become unpredictable after a few dozen collisions. Eventually the balls would degenerate into a gas, spread throughout the apparatus, with a Maxwell distribution of velocities. Even if classical balls could be shot with perfect accuracy into a perfect apparatus, fluctuating tidal forces from turbulence in the atmospheres of nearby stars would be enough to randomize their motion within a few hundred collisions. Needless to say, the trajectory would be spoiled much sooner if stronger nearby noise sources (e.g., thermal radiation and conduction) were not eliminated.

Practically, this dynamical instability means that the balls' velocities and positions would have to be corrected after every few collisions. The resulting computer, although no longer thermodynamically reversible, might

still be of some practical interest, since energy cost per step of restoring the trajectory might be far less than the kinetic energy accounting for the computation's speed.

One way of making the trajectory insensitive to noise would be to use square balls, holonomically constrained to remain always parallel to each other and to the fixed walls. Errors would then no longer grow exponentially, and small perturbations could simply be ignored. Although this system is consistent with the laws of classical mechanics it is a bit unnatural, since there are no square atoms in nature. A macroscopic square particle would not do, because a fraction of its kinetic energy would be converted into heat at each collision. On the other hand, a square molecule might work, if it were stiff enough to require considerably more than kT of energy to excite it out of its vibrational ground state. To prevent the molecule from rotating, it might be aligned in an external field strong enough to make the energy of the first librational excited state similarly greater than kT. One would still have to worry about losses when the molecules collided with the mirrors. A molecule scattering off even a very stiff crystal has sizable probability of exciting long-wavelength phonons, thereby transferring energy as well as momentum. This loss could be minimized by reflecting the particles from the mirrors by long-range electrostatic repulsion, but that would interfere with the use of short-range forces for collisions between molecules, not to mention spoiling the uniform electric field used to align the molecules.

Although quantum effects might possibly help stabilize a ballistic computer against external noise, they introduce a new source of internal instability in the form of wave-packet spreading. Benioff's discussion (1982) of quantum ballistic models shows how wave packet spreading can be prevented by employing a periodically varying Hamiltonian, but not apparently by any reasonably simple time-independent Hamiltonian.

In summary, although ballistic computation is consistent with the laws of classical and quantum mechanics, there is no evident way to prevent the signals' kinetic energy from spreading into the computer's other degrees of freedom. If this spread is combatted by restoring the signals, the computer becomes dissipative; if it is allowed to proceed unchecked, the initially ballistic trajectory degenerates into random Brownian motion.

3. BROWNIAN COMPUTERS

If thermal randomization of the kinetic energy cannot be avoided, perhaps it can be exploited. Another family of models may be called Brownian computers, because they allow thermal noise to influence the

trajectory so strongly that all moving parts have nearly Maxwellian velocities, and the trajectory becomes a random walk. Despite this lack of discipline, the Brownian computer can still perform useful computations because its parts interlock in such a way as to create a labyrinth in configuration space, isomorphic to the desired computation, from which the trajectory is prevented from escaping by high-potential-energy barriers on all sides. Within this labyrinth the system executes a random walk, with a slight drift velocity in the intended direction of forward computation imparted by coupling the system to a weak external driving force.

In more concrete terms, the Brownian computer makes logical state transitions only as the accidental result of the random thermal jiggling of its information-bearing parts, and is about as likely to proceed backward along the computation path, undoing the most recent transition, as to proceed forward. The chaotic, asynchronous operation of a Brownian computer is unlike anything in the macroscopic world, and it may at first appear inconceivable that such an apparatus could work; however, this style of operation is quite common in the microscopic world of chemical reactions, were the trial and error action of Brownian motion suffices to bring reactant molecules into contact, orient and bend them into a specific conformation ("transition state") that may be required for reaction, and separate the product molecules after reaction. It is well known that all chemical reactions are in principle reversible: the same Brownian motion that accomplishes the forward reaction also sometimes brings product molecules together, pushes them backward through the transition state, and lets them emerge as reactant molecules. Though Brownian motion is scarcely noticeable in macroscopic bodies (e.g., $(kT/m)^{1/2} \approx 10^{-6}$ cm/sec for a 1-g mass at room temperature), it enables even rather large molecules, in a fraction of a second, to accomplish quite complicated chemical reactions, involving a great deal of trial and error and the surmounting of potential energy barriers of several kT in order to arrive at the transition state. On the other hand, potential energy barriers of order 100 kT, the typical strength of covalent bonds, effectively obstruct chemical reactions. Such barriers, for example, prevent DNA from undergoing random rearrangements of its base sequence at room temperature.

To see how a molecular Brownian computer might work, we first consider a simpler apparatus: a Brownian tape-copying machine. Such an apparatus already exists in nature, in the form of RNA polymerase, the enzyme that synthesizes a complementary RNA copy of one or more genes of a DNA molecule. The RNA then serves to direct the synthesis of the proteins encoded by those genes (Watson, 1970). A schematic snapshot of RNA polymerase in action is given in Figure 3. In each cycle of operation, the enzyme takes a small molecule (one of the four nucleotide pyrophos-

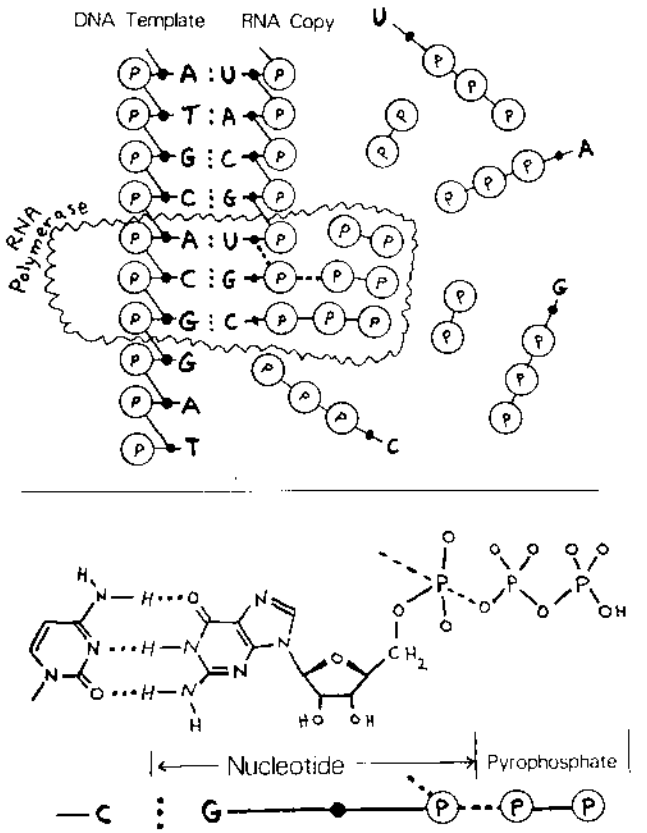

Fig. 3. RNA Polymerase synthesizing a complementary RNA strand on a single-strand DNA "template." Double and triple dots between DNA and RNA bases indicate base-pairing interaction; dashed lines indicate covalent bonds being formed and broken by RNA polymerase. Below, in more detail, the arriving GTP monomer about to lose its pyrophosphate group and be attached to the growing RNA strand.

phates, ATP, GTP, CTP, or UTP, whose base is complementary to the base about to be copied on the DNA strand) from the surrounding solution, forms a covalent bond between the nucleotide part of the small molecule and the existing uncompleted RNA strand, and releases the pyrophosphate part into the surrounding solution as a free pyrophosphate molecule (PP). The enzyme then shifts forward one notch along the DNA in preparation for copying the next nucleotide. In the absence of the enzyme, this reaction would occur with a negligible rate and with very poor specificity for selecting bases correctly complementary to those on the DNA strand. Assuming RNA polymerase to be similar to other enzymes whose mechanisms have been studied in detail, the enzyme works by forming many weak (e.g., van der Waals and hydrogen) bonds to the DNA, RNA, and incoming nucleotide pyrophosphate, in such a way that if the incoming nucleotide is correctly base-paired with the DNA, it is held in the correct transition state

conformation for forming a covalent bond to the end of the RNA strand, while breaking the covalent bond to its own pyrophosphate group. The transition state is presumably further stabilized (its potential energy lowered) by favorable electrostatic interaction with strategically placed charged groups on the enzyme.

The reaction catalyzed by RNA polymerase is reversible: sometimes the enzyme takes up a free pyrophosphate molecule, combines it with the end nucleotide of the RNA, and releases the resulting nucleotide pyrophosphate into the surrounding solution, meanwhile backing up one notch along the DNA strand. The operation of the enzyme thus resembles a one-dimensional random walk (Figure 4), in which both forward and backward steps are possible, and would indeed occur equally often at equilibrium. Under biological conditions, RNA polymerase is kept away from equilibrium by other metabolic processes, which continually supply ATP, GTP, UTP, and CTP and remove PP, thereby driving the chain of reactions strongly in the direction of RNA synthesis. In domesticated bacteria, RNA polymerase runs forward at about 30 nucleotides per second, dissipating about $20kT$ per nucleotide, and making less than one mistake per ten thousand nucleotides.

In the laboratory, the speed and direction of operation of RNA polymerase can be varied by adjusting the reactant concentrations. The closer these are to equilibrium, the slower and the less dissipatively the enzyme works. For example, if ATP, GTP, UTP, and CTP were each present in 10% excess over the concentration that would be in equilibrium with a given ambient PP concentration, RNA synthesis would drift slowly forward, the enzyme on average making 11 forward steps for each 10 backward steps. These backward steps do not constitute errors, since they are undone by subsequent forward steps. The energy dissipation would be $kT\ln(11/10)\approx 0.1kT$ per (net) forward step, the difference in chemical potential between reactants and products under the given conditions. More

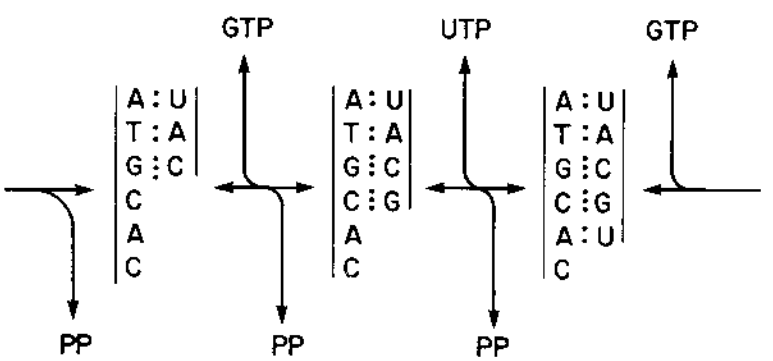

Fig. 4. RNA polymerase reaction viewed as a one-dimensional random walk.

generally, a dissipation of ϵ per step results in forward/backward step ratio of $e^{+\epsilon/kT}$, and for small ϵ, a net copying speed proportional to ϵ.

The analysis so far has ignored true errors, due to uncatalyzed reactions. Because these occur in some fixed, hardware-dependent ratio η_0 to the gross (rather than the net) number of catalyzed transitions, they set a limit on how slowly the copying system can be driven and still achieve reasonable accuracy. For example, if a copying system with an intrinsic error rate of 10^{-4} were driven at $0.1kT$ per step, its error rate would be about 10^{-3}; but if it were driven at $10^{-4}kT$ or less, near total infidelity would result. Because the intrinsic error rate is determined by the difference in barriers opposing correct and incorrect transitions, it is a function of the particular chemical hardware, and does not represent a fundamental thermodynamic limit. In principle it can be made arbitrarily small by increasing the size and complexity of the recognition sites (to increase the potential energy difference ΔE between correct and incorrect reaction paths), by lowering the temperature (to increase the Boltzmann ratio $e^{\Delta E/kT}$ of correct to error transitions without changing ΔE) and by making the apparatus larger and more massive (to reduce tunneling). In situations calling for very high accuracy (e.g., DNA copying), the genetic apparatus apparently uses another strategem for reducing errors: dissipative error correction or proofreading (Hopfield, 1974; Ninio, 1975), depicted in Figure 5. The dissipation-error tradeoff for model nonproofreading and proofreading copying systems is discussed by Bennett (1979). An amusing if impractical feature of this tradeoff is that when a copying system is operated at very low speed (and therefore high error rate), the errors themselves serve as a thermodynamic driving force, and can push the copying slowly forward even in the presence of a small reverse bias in the driving reaction. Of course, in obediance to the second law, the entropy of the incorporated errors more than makes up for the work done against the external reactants.

A true chemical Turing machine is not difficult to imagine (Figure 6). The tape might be a linear informational macromolecule analogous to RNA, with an additional chemical group attached at one site to encode the head state (α) and location. Several hypothetical enzymes (one for each of the Turing machine's transition rules) would catalyze reactions of the macromolecule with small molecules in the surrounding solution, transforming the macromolecule into its logical successor. The transition $\alpha 0 \rightarrow 1R\beta$, for example, would be carried out by an enzyme that brings with it the groups 1 and β that must be added during the transition, and has additional specific affinities allowing it to temporarily bind to groups 0 and α that must be removed. (Real enzymes with multiple, highly specific binding sites are well known, e.g., the acylating enzymes of protein synthesis.) In Figure 6, the hypothetical enzyme binds on the right, since its transition rule calls

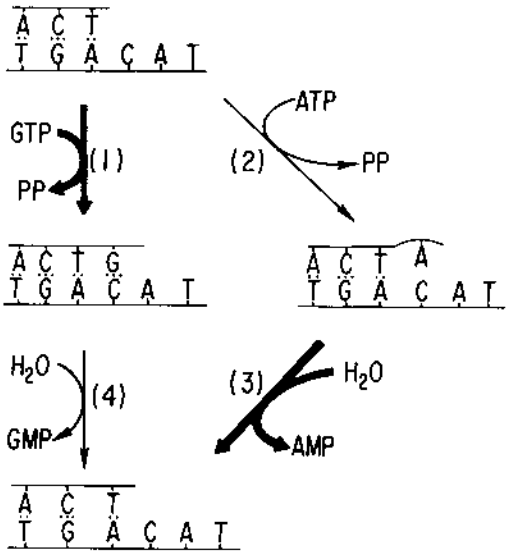

Fig. 5. Proofreading in DNA replication. The enzyme responsible for copying DNA usually inserts the correct nucleotide (1), but occasionally inserts an incorrect one (2). To counter this, another enzyme (or another active site on the same enzyme) catalyzes a proofreading reaction (3), which preferentially removes incorrectly paired nucleotides from the end of an uncompleted DNA strand. The proofreading enzyme also occasionally makes mistakes, removing a nucleotide even though it is correct (4). After either a correct or an incorrect nucleotide has been removed, the copying enzyme gets another chance to try to insert it correctly, and the proofreading enzyme gets another chance to proofread, etc. It is important to note that the proofreading reaction is not simply the thermodynamic reverse of the copying reaction: it uses different reactants, and has a different specificity (favoring incorrect nucleotides, while the copying reaction favors correct nucleotides). The minimum error rate (equal to the product of the error rates of the writing and proofreading steps) is obtained when both reactions are driven strongly forward, as they are under physiological conditions. Proofreading is an interesting example of the use of thermodynamic irreversibility to perform the logically irreversible operation of error correction.

for a right shift. After the requisite changes have been made, it drops off and is readied for reuse. At some point in their cycle of use the hypothetical enzymes are made to catalyze a reaction involving external reactants (here ATP:ADP), whose concentrations can be adjusted to provide a variable driving force.

Assume for the moment that the enzymatic Turing machine is *logically reversible*, i.e., that no whole-machine state has more than one logical predecessor (this mathematical requirement on the structure of the computation will be discussed in the next section). Then the net computation speed will be linear in the driving force, as was the case with RNA polymerase, because the logically accessible states (i.e., configurations of the macromolecule accessible by forward and backward operation of the enzymes) form a one-dimensional chain, along which the executes a random walk with drift velocity proportional to ϵ/kT.

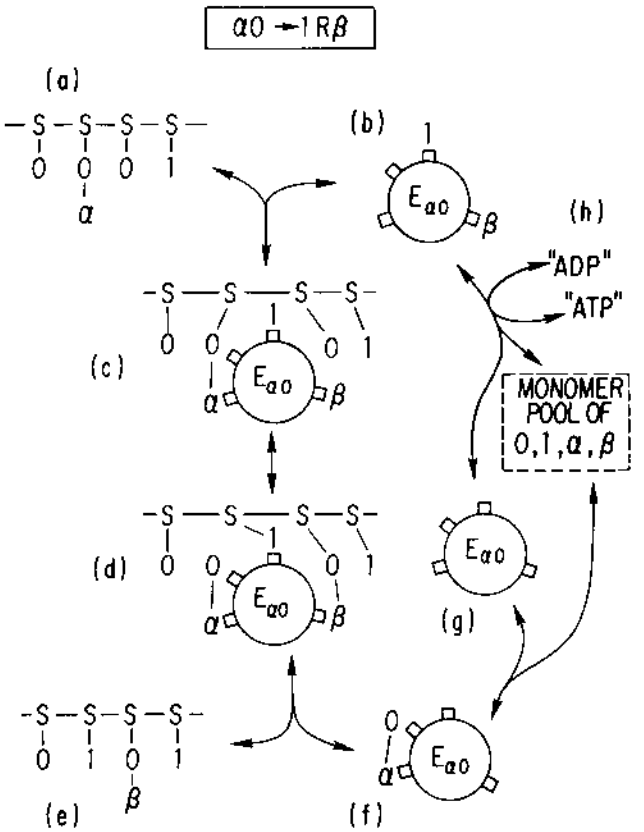

Fig. 6. Hypothetical enzymatic Turing machine. Macromolecular tape (a) consists of a structural backbone S–S–S bearing tape symbols 1,0 and head marker α. Macromolecule reacts (c,d) with enzyme (b) that catalyzes the transition $\alpha 0 \rightarrow 1R\beta$, via specific binding sites (tabs), thereby resulting in logical successor configuration (e). Enzyme is then prepared for reuse (f,g,h). Coupling to external reaction (h) drives the reactions, which would otherwise drift indifferently forward and backward, in the intended forward direction.

It is also possible to imagine an error-free Brownian Turing machine made of rigid, frictionless clockwork. This model (Figure 7) lies between the billiard-ball computer and the enzymatic computer in realism because, on the one hand, no material body is perfectly hard; but on the other hand, the clockwork model's parts need not be machined perfectly, they may be fit together with some backlash, and they will function reliably even in the presence of environmental noise. A similar model has been considered by Reif (1979) in connection with the **P = PSPACE** question in computational complexity.

The baroque appearance of the clockwork Turing machine reflects the need to make all its parts interlock in such a way that, although they are free to jiggle locally at all times, no part can move an appreciable distance except when it is supposed to be making a logical transition. In this respect it resembles a well worn one of those wooden cube puzzles that must be

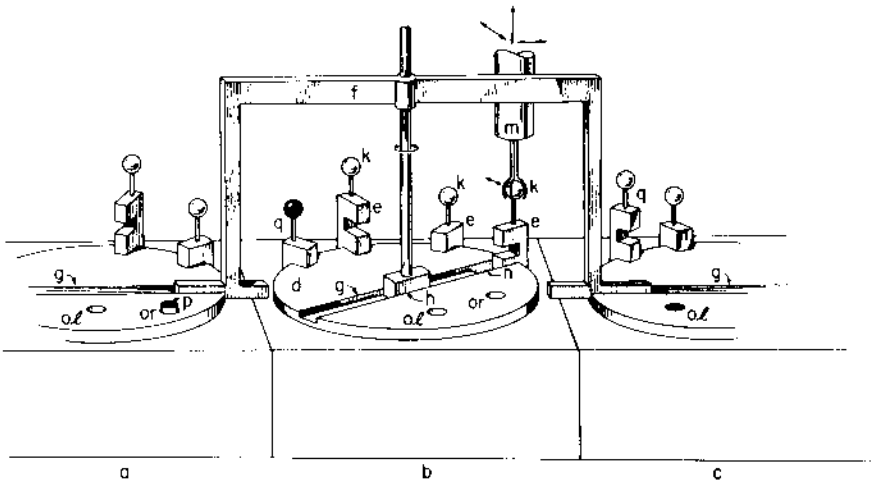

Fig. 7. Brownian Turing machine made of rigid, frictionless, loosely fitting clockwork. This figure shows the Turing machine tape (a,b,c) and the read–write–shift equipment. The machine is scanning square b. Each tape square has a disk (d) which interlocks with several E-shaped bit-storage blocks (e), holding them in the up (1) or down (0) position. A framework (f) fits over the scanned tape square, engaging the disks of the two adjacent squares (via their grooves g), to keep them from rotating when they are not supposed to. After the bits are read (cf. next figure) they must in general be changed. In order to change a bit, its knob (k) is first grasped by the manipulator (m), then the notch (n) is rotated into alignment by the screwdriver (h) and the bit storage block (e) is slid up or down. The block is then locked into place by further rotating the disk, after which the manipulator can safely let go and proceed to grasp the next bit's knob. Each tape square has a special knob (q) that is used to help constrain the disks on nonscanned tape squares. In principle these might all be constrained by the framework (f), but that would require making it infinitely large and aligning it with perfect angular accuracy. To avoid this, the framework (f) is used only to constrain the two adjacent tape squares. All the remaining tape squares are indirectly constrained by pegs (p) coupled to the special knob (q) of an adjacent square. The coupling (a lever arrangement hidden under the disk) is such that, when any square's q knob is down, a peg (p) engages the rightmost of two openings (o r) on the next tape square to the left, and another peg disengages the leftmost (o l) of two openings on the next tape square to the right. A q knob in the up position does the opposite: it frees the tape square to its left and locks the tape square to its right. To provide an outward-propagating chain of constraints on each side of the scanned square, all the q knobs to its right must be up, and all the q knobs to its left must be down. The q knob on the scanned square can be in either position, but just before a right shift it is lowered, and just before a left shift it is raised. To perform the shift, the screwdriver rotates the scanned square's groove (g) into alignment with the framework, then the manipulator (m), by grasping some convenient knob, pulls the whole head apparatus (including m itself, as well as f, h, and parts not shown) one square to the left or right.

solved by moving one part a small distance, which allows another to move in such a way as to free a third part, etc. The design differs from conventional clockwork in that parts are held in place only by the hard constraints of their loosely fitting neighbors, never by friction or by spring pressure. Therefore, when any part is about to be moved (e.g., when one of the E-shaped blocks used to store a bit of information is moved by the manipulator **m**), it must be grasped in its current position before local constraints on its motion are removed, and, after the move, local constraints must be reimposed before the part can safely be let go of.

Perhaps the most noteworthy feature of the machine's operation is the "obstructive read" depicted in Figure 8. In general, the coordinated motion of the screwdriver **h** and manipulator **m**, by which the Turing machine control unit acts on the tape, can be described as a deterministic unbranched path in the five-dimensional configuration space spanning the screwdriver's rotation and the manipulator's translation and grasp (because of backlash, this path is not single trajectory, but a zero-potential-energy channel of finite width surrounded by infinite potential energy barriers, within which the system performs a Brownian random walk, with a slight forward drift velocity). However, before it can know what to write or which way to shift, the control unit must ascertain the current contents of the scanned tape square. To do this, during the read stage of the machine cycle, the path of the manipulator branches nondeterministically into two paths, one of which is obstructed (due to collision with the knob **k**) by a bit in the up position, the other by the same bit in the down position. This bifurcation followed by obstruction is repeated for each additional bit stored on the tape square, so that, by the time the manipulator has negotiated all the

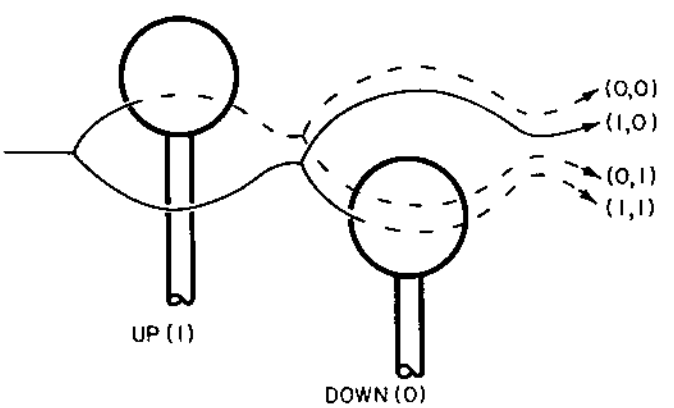

Fig. 8. Obstructive read. The clockwork Turing machine's control unit guides the manipulator along a branching path, one of whose branches is obstructed by a knob in the up position, the other by the same knob in the down position.

bifurcations and obstructions, it is again on a single path determined by the contents of the scanned tape square. If the manipulator by chance wanders into the wrong path at a bifurcation and encounters an obstruction, the forward progress of the computation is delayed until Brownian motion jiggles the manipulator back to the bifurcation and forward again along the right path.

Figure 9 suggests how the manipulator and screwdriver might be driven through their paces by the main control unit. A master camshaft, similar to a stereophonic phonograph record, would contain a network of tunnels isomorphic with the Turing machine's finite state transition graph. A weak spring (the only spring in the whole apparatus) biases the camshaft's Brownian motion, causing it to revolve on average in the direction corresponding to forward computation. Revolution of the camshaft imposes certain motions on a captive cam follower, which in turn are translated by appropriate mechanical linkages into synchronous motions of the manipulator and screwdriver in Figure 7, required to perform read, write, and shift operations on the Turing machine tape. During the read phase, the cam follower passes through a bifurcation in its tunnel for each bit to be read, causing the manipulator to perform an obstructive read, and delaying the

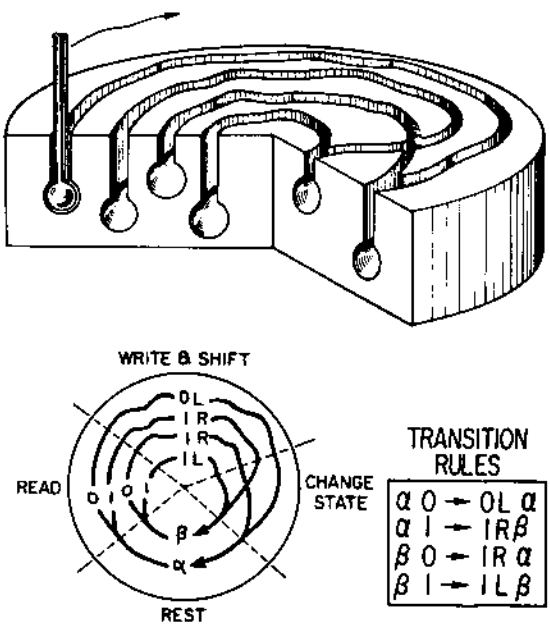

Fig. 9. Above: master camshaft of the clockwork Turing machine's control unit. Below: top view of camshaft, and the transition rules to which its tunnels are isomorphic.

computation until the cam follower, by Brownian trial and error, chooses the right tunnel. The obstructive read places the cam follower at the beginning of a specific tunnel segment corresponding to the current head state and tape symbol, and the rest of this tunnel segment drives the manipulator and screwdriver through the coordinated motions necessary to accomplish the write and shift operations. The obstructive read also serves another, less obvious purpose: it prevents the machine from wandering backward into states that are not logical predecessors of the present state. If the machine is logically reversible as supposed, this means that the only unobstructed path for the camshaft to rotate backwards one full revolution is the path leading to the present state's unique logical predecessor.

As in the case of the enzymatic Turing machine, the drift velocity is linear in the dissipation per step. Because the clockwork Turing machine cannot make illogical transitions, the only kind of error it is susceptible to is failure to be in the final logical state of its computation. Indeed, if the driving force ϵ is less than kT, any Brownian computer will at equilibrium spend most of its time in the last few predecessors of the final state, spending only about ϵ/kT of its time in the final state itself. However, the final state occupation probability can be made arbitrarily large, independent of the number of steps in the computation, by dissipating a little extra energy during the final step, a "latching energy" of $kT\ln(kT/\epsilon)+kT\ln(1/\eta)$ sufficing to raise the equilibrium final state occupation probability to $1-\eta$.

Quantum mechanics probably does not have a major qualitative effect on Brownian computers: in an enzymatic computer, tunneling and zero-point effects would modify transition rates for both catalyzed and uncatalyzed reactions; a quantum clockwork computer could be viewed as a particle propagating in a multidimensional labyrinth in configuration space (since the clockwork computer's parts are assumed to be perfectly hard, the wave function could not escape from this labyrinth by tunneling). Both models would exhibit the same sort of diffusive behavior as their classical versions.

[It should perhaps be remarked that, although energy transfers between a quantum system and its environment occur via quanta (e.g., photons of black body radiation) of typical magnitude about kT, this fact does not by itself imply any corresponding coarseness in the energy cost per step: a net energy transfer of $0.01kT$ between a Brownian computer and its environment could for example be achieved by emitting a thermal photon of $1.00kT$ and absorbing one of $0.99kT$. The only limitation on this kind of spontaneous thermal fine tuning of a quantum system's energy comes from its energy level spacing, which is less than kT except for systems so cold that the system as a whole is frozen into its quantum ground state.]

4. LOGICAL REVERSIBILITY

Both the ballistic and Brownian computers require a change in programming style: *logically irreversible* operations (Landauer, 1961) such as erasure, which throw away information about the computer's preceding logical state, must be avoided. These operations are quite numerous in computer programs as ordinarily written; besides erasure, they include overwriting of data by other data, and entry into a portion of the program addressed by several different transfer instructions. In the case of Turing machines, although the individual transition rules (quintuples) are reversible, they often have overlapping ranges, so that from a given instantaneous description it is not generally possible to infer the immediately preceding instantaneous description (Figure 1). In the case of combinational logic, the very gates out of which logic functions are traditionally constructed are for the most part logically irreversible (e.g., AND, OR, NAND), though NOT is reversible.

Logically irreversible operations must be avoided entirely in a ballistic computer, and for a very simple reason: the merging of two trajectories into one cannot be brought about by conservative forces. In a Brownian computer, a small amount of logical irreversibility can be tolerated (Figure 10), but a large amount will greatly retard the computation or cause it to fail completely, unless a finite driving force (approximately $kT \ln 2$ per bit of information thrown away) is applied to combat the computer's tendency to drift backward into extraneous branches of the computation. Thus driven, the Brownian computer is no longer thermodynamically reversible, since its dissipation per step no longer approaches zero in the limit of zero speed.

In spite of their ubiquity, logically irreversible operations can be avoided without seriously limiting the power of computers. A means of simulating arbitrary irreversible computations reversibly is given by Bennett (1973) using Turing machines, was independently discovered by Fredkin, using reversible Boolean logic (Toffoli, 1980), and is outlined below.

We begin by noting that it is easy to render any computer reversible in a rather trivial sense, by having it save all the information it would otherwise have thrown away. For example the computer could be given an extra "history" tape, initially blank, on which to record enough about each transition (e.g., for a Turing machine, which quintuple was being used) that the preceding state would be uniquely determined by the present state and the last record on the history tape. From a practical viewpoint this does not look like much of an improvement, since the computer has only postponed the problem of throwing away unwanted information by using the extra tape as a garbage dump. To be usefully reversible, a computer ought to be required to clean up after itself, so that at the end of the computation the

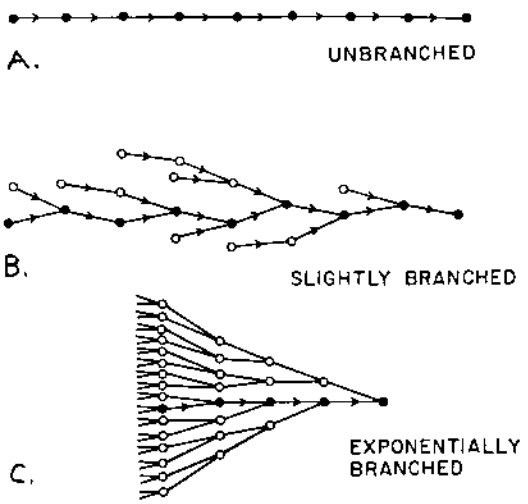

Fig. 10. Various kinds of computation graph, with black nodes denoting logical states (instantaneous descriptions, in Turing machine terminology) on the intended computation path, and open nodes denoting extraneous predecessors. Arrows indicate intended direction of transitions, but if the driving force is weak, backward transitions occur nearly as often as forward ones. In a strictly reversible computation (A), the graph is unbranched, and an arbitrarily small driving force ϵ suffices to drive the computation forward with drift velocity proportional to ϵ. An arbitrarily small driving force still suffices for a slightly branched graph (B), with a few extraneous predecessors per state on the intended path, but the computation proceeds more slowly, due to temporary detours onto the extraneous branches. Slightly branching trees occur, for example when the same variable is assigned several times in succession: at any point in the computation, the most recent assignment can be randomly undone (giving the variable any value at all), but the result is typically a "garden-of-Eden" state with no predecessor, because this random value is inconsistent with the forward result of the previous assignment. Exponentially branching trees (C) also occur, for example, in loops that assign a succession of different variables. If such a tree is infinite, then a small driving force is insufficient to keep the computation from wandering onto an extraneous branch never to return; to drive the computation forward in such a tree, the dissipation per step must exceed kT times the mean number of immediate predecessors per state. Even if the exponential backward branching is eventually stopped by garden-of-Eden states, as is commonly the case, extraneous states may outnumber states on the intended path by factors of 2^{100} or so, slowing down the computation by that much unless backward excursions are again suppressed by dissipating about kT times the logarithm of the mean number of immediate predecessors of states near the intended computation path.

only data remaining are the desired output and the originally furnished input. [A general-purpose reversible computer must be allowed to save its input, or some equivalent information; otherwise it could not perform computations in which the input was not uniquely determined by the output. Formally this amounts to embedding the partial recursive function $x \to \varphi(x)$, computed by the original irreversible machine, in a 1 : 1 partial recursive function $x \to \langle x, \varphi(x) \rangle$, to be computed by the reversible machine;

since 1 : 1 functions are the only kind that can be computed by reversible machines.]

A tape full of random data can only be erased by an irreversible process. However, the history produced by the above untidy reversible computer is not random, and it can be gotten rid of reversibly, by exploiting the redundancy between it and the computation that produced it. If, at the end of the untidy computer's computation, another stage of computation were begun, using the inverse of the untidy machine's transition function, the inverse, or "cleanup" machine would begin undoing the untidy machine's computation step by step, eventually returning the history tape to its original blank condition. Since the untidy machine is reversible and deterministic, its cleanup machine is also reversible and deterministic. [Bennett (1973) gives detailed constructions of the reversible untidy and cleanup machines for an arbitrary irreversible Turing machine.] The cleanup machine, of course, is not a general-purpose garbage disposer: the only garbage it can erase is the garbage produced by the untidy machine of which it is the inverse.

Putting the untidy and cleanup machines together, one obtains a machine that reversibly does, then undoes, the original irreversible computation. This machine is still useless because the desired output, produced during the untidy first stage of computation, will have been eaten up along with the undesired history during the second cleanup stage, leaving behind only a reconstructed copy of the original input. Destruction of the desired output can be easily prevented, by introducing still another stage of computation, after the untidy stage but before the cleanup stage. During this stage the computer makes an extra copy of the output on a separate, formerly blank tape. No additional history is recorded during this stage, and none needs to be, since copying onto blank tape is already a 1 : 1 operation. The subsequent cleanup stage therefore destroys only the original of the output but not the copy. At the end of its three-stage computation, the computer contains the (reconstructed) original input plus the intact copy of the output. All other storage will have been restored to its original blank condition. Even though no history remains, the computation is reversible and deterministic, because each of its stages has been so. The use of separate tapes for output and history is not necessary; blank portions of the work tape may be used instead, at the cost of making the simulation run more slowly (quadratic rather than linear time). Figure 11A summarizes the three-stage computation.

The argument just outlined shows how an arbitrary Turing machine can be simulated in not much more time by a reversible Turing machine, at the cost of including the input as part of the output, and perhaps using a lot of temporary storage for the history. By cleaning up the history more often

A

Computation Stage	Contents of Work area	History area	Output area
	_INPUT	_	_
Untidy	WORK	HISTO_	_
	_OUTPUT	HISTORY_	_
	OUTPUT	HISTORY_	OUT_
Copy output	OUTPUT_	HISTORY_	OUTPUT_
	_OUTPUT	HISTORY_	_OUTPUT
Cleanup	WORK	HISTO_	_OUTPUT
	_INPUT	_	_OUTPUT

B

Computation Stage	Contents of Work area	Hist. area	Output area
	INPUT	-	-
1. Untidy φ comp.	OUTPUT	φ HISTORY	-
2. Copy output	OUTPUT	φ HISTORY	OUTPUT
3. Cleanup φ comp.	INPUT	-	OUTPUT
4. Interchange data	OUTPUT	-	INPUT
5. Untidy φ^{-1} comp.	INPUT	φ^{-1} HISTORY	INPUT
6. Cancel extra input	INPUT	φ^{-1} HISTORY	-
7. Cleanup φ^{-1} comp.	OUTPUT	-	-

Fig. 11. (A) Reversible simulation of an irreversible computation. The reversible computer has three storage areas (e.g., tapes): a work area in which the simulation takes place; a history area in which garbage generated by the irreversible computation is saved temporarily, thereby rendering the computation reversible; and an output area. The work area initially contains the input; the history and output areas are initially blank. The computation takes place in three stages, representative snapshots of which are shown. The underbars represent the locations of read/write heads for a three-tape machine, or analogous place markers for other machines. (B) Reversible computation of an arbitrary 1 : 1 function φ with no extra output, given irreversible algorithms for φ and φ^{-1}. The computation proceeds by seven stages as shown. Stage 5 has the sole purpose of producing the φ^{-1} history, which, after the extra input has been reversibly erased in stage 6, serves in stage 7 to destroy itself and the remaining copy of the input, leaving only the desired output.

(Bennett, 1973), space on the history tape may be traded off against time or additional garbage output. This tradeoff, the details of which remain to be worked out, provides an upper bound on the cost of making computations reversible. However, it has been observed in practice that many computations (e.g., numerical integration of differential equations) can be performed by reversible algorithms with *no* penalty in time, storage, or extra output.

In cases where the original irreversible Turing machine computes a 1 : 1 function, it can be simulated reversibly with no additional output, but with perhaps an exponential increase in run time. This is done by combining McCarthy's (1956) trial-and-error procedure for effectively computing the inverse of a 1 : 1 partial recursive function [given an algorithm φ for the original function and an argument y, $\varphi^{-1}(y)$ is defined as the first element of the first ordered pair $\langle x, s \rangle$ such that $\varphi(x) = y$ in less than s steps] with Bennett's procedure (1973; Figure 11B] for synthesizing a reversible Turing machine with no extra output from two mutually inverse irreversible Turing machines. These results imply that the reversible Turing machines provide a Gödel numbering of 1 : 1 partial recursive functions (i.e., every 1 : 1 partially reversible function is computable by a reversible TM and vice versa). The construction of a reversible machine from an irreversible machine and its inverse implies that the open question, of whether there exists a 1 : 1 function much easier to compute by an irreversible machine than by any reversible machine, is equivalent to the question of whether there is an easy 1 : 1 function with a hard inverse.

Simulation of irreversible logic functions by reversible gates is analogous (Toffoli, 1980) to the constructions for Turing machines. The desired function is first embedded in a one-to-one function (e.g., by extending the output to include a copy of the input) which is then computed by reversible gates, such as the three-input, three-output AND/NAND gate. The first two inputs of this gate simply pass through unchanged to become the first two outputs; the third output is obtained by taking the EXCLUSIVE-OR of the third input with the AND of the first two inputs. The resulting mapping from three bits onto three bits (which is its own inverse) can be used together with constant inputs to compute any logic function computable by conventional gates, but in so doing, may produce extra "garbage" bits analogous to the reversible Turing machine's history. If the function being computed is 1 : 1, these bits need not appear in the final output, because they can be disposed of by a process analogous to the reversible Turing machine's cleanup stage, e.g., by feeding them into a mirror image of the reversible logic net that produced them in the first place. The "interaction gate" used in the ballistic computer (Fredkin and Toffoli, 1981) may be regarded as having two inputs and four outputs (respectively, x & y, y & $\neg$ x, x & $\neg$ y, and x & y) for the four possible exit paths from the collision

zone between balls x and y. The interaction gate is about as simple as can be imagined in its physical realization, yet it suffices (with the help of mirrors to redirect and synchronize the balls) to synthesize all conservative Boolean functions, within which all Boolean functions can be easily embedded. The rather late realization that logical irreversibility is not an essential feature of computation is probably due in part to the fact that reversible gates require somewhat more inputs and outputs (e.g., 3:3 or 2:4) to provide a basis for nontrivial computation than irreversible gates do (e.g., 2:1 for NAND).

5. REVERSIBLE MEASUREMENT AND MAXWELL'S DEMON

This section further explores the relation between logical and thermodynamic irreversibility and points out the connection between logical irreversibility and the Maxwell's demon problem.

Various forms of Maxwell's demon have been described; a typical one would be an organism or apparatus that, by opening a door between two equal gas cylinders whenever a molecule approached from the right, and closing it whenever a molecule approached from the left, would effortlessly concentrate all the gas on the left, thereby reducing the gas's entropy by $Nk\ln 2$. The second law forbids any apparatus from doing this reliably, even for a gas consisting of a single molecule, without producing a corresponding entropy increase elsewhere in the universe.

It is often supposed that *measurement* (e.g., the measurement the demon must make to determine whether the molecule is approaching from the left or the right) is an unavoidably irreversible act, requiring an entropy generation of at least $k\ln 2$ per bit of information obtained, and that this is what prevents the demon from violating the second law. In fact, as will be shown below, measurements of the sort required by Maxwell's demon can be made reversibly, provided the measuring apparatus (e.g., the demon's internal mechanism) is in a standard state before the measurement, so that measurement, like the copying of a bit onto previously blank tape, does not overwrite information previously stored there. Under these conditions, the essential irreversible act, which prevents the demon from violating the second law, is not the measurement itself but rather the subsequent restoration of the measuring apparatus to a standard state in preparation for the next measurement. This forgetting of a previous logical state, like the erasure or overwriting of a bit of intermediate data generated in the course of a computation, entails a many-to-one mapping of the demon's physical state, which cannot be accomplished without a corresponding entropy increase elsewhere.

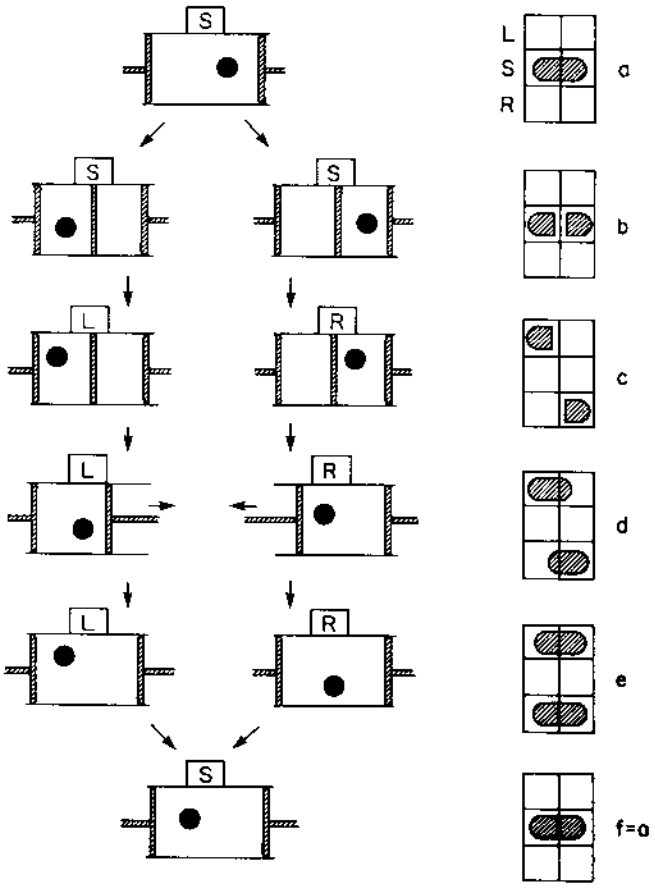

Fig. 12. A one-molecule Maxwell's demon apparatus.

Figure 12 shows the cycle of operation of a one-molecule Maxwell's demon apparatus. The left side of the figure shows the apparatus, and the right side shows the sequence changes in its phase space, depicted schematically as a product of a horizontal coordinate representing the location of the molecule and a vertical coordinate representing the physical state of the demon's "mind." The demon's mind has three states: its standard state S before a measurement, and two states **L** and **R** denoting the result of a measurement in which the molecule has been found on the left or right, respectively. At first (a) the molecule wanders freely throughout the apparatus and the demon is in the standard state S, indicating that it does not know where the molecule is. In (b) the demon has inserted a thin partition trapping the molecule on one side or the other. Next the demon performs a reversible measurement to learn (c) whether the molecule is on the left or the right. The demon then uses this information to extract $kT\ln 2$ of isothermal work from the molecule, by inserting a piston on the side not containing the molecule and allowing the molecule to expand (d) against the

piston to fill the whole apparatus again (e). Notice that a different manipulation is required to extract work from the molecule depending on which side it is on; this is why the demon must make a measurement, and why at (d) the demon will be in one of two distinct parts of its own phase space depending on the result of that measurement. At (e) the molecule again fills the whole apparatus and the piston is in its original position. The only record of which side the molecule came from is the demon's record of the measurement, which must be erased to put the demon back into a standard state. This erasure (e–f) entails a twofold compression of the occupied volume of the demon's phase space, and therefore cannot be made to occur spontaneously except in conjunction with a corresponding entropy increase elsewhere. In other words, all the work obtained by letting the molecule expand in stage (d) must be converted into heat again in order to compress the demon's mind back into its standard state.

In the case of a measuring apparatus which is *not* in a standard state before the measurement, the compression of the demon's state, and the compensating entropy increase elsewhere, occur at the same time as the measurement. However, I feel it important even in this case to attribute the entropy cost to logical irreversibility, rather than to measurement, because in doing the latter one is apt to jump to the erroneous conclusion that all transfers of information, e.g., the synthesis of RNA, or reversible copying onto blank tape, have an irreducible entropy cost of order $kT\ln 2$ per bit.

As a further example of reversible copying and measurement, it may be instructive to consider a system simpler and more familiar than RNA polymerase, viz., a one-bit memory element consisting of a Brownian particle in a potential well that can be continuously modulated between bistability (two minima separated by a barrier considerably higher than kT) and monostability (one minimum), as well as being able to be biased to favor one well or the other. Such systems have been analyzed in detail in Landauer (1961) and Keyes and Landauer (1970); a physical example (Figure 13) would be an ellipsoidal piece of ferromagnetic material so small that in the absence of a field it consists of a single domain, magnetized parallel or antiparallel to the ellipse axis (alternatively, a round piece of magnetically anisotropic material could be used). Such a system can be modulated between bistability and monostability by a transverse magnetic field, and biased in favor of one minimum or the other by a longitudinal magnetic field. When the transverse field just barely abolishes the central minimum, the longitudinal magnetization becomes a "soft mode," very sensitive to small longitudinal components of the magnetic field.

This sensitivity can be exploited to copy information reversibly from one memory element to another, provided the memory element destined to

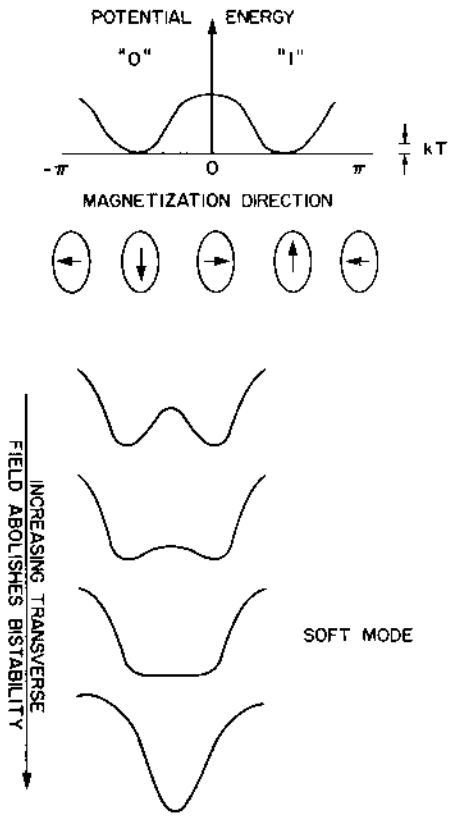

Fig. 13. A bistable potential well, realized by a one-domain ferromagnet. Increasing transverse magnetic field abolishes the bistability, resulting in a "soft mode," very sensitive to small longitudinal fields.

receive the information is in a standard state initially, so that the copying is logically reversible. Figure 14 shows how this reversible copying might be done. The apparatus contains two fixed memory elements, one (top) containing a zero for reference and the other (bottom) containing the data bit to be copied. A third memory element is movable and will be used to receive the copy. It is initially located next to the reference element and also contains a zero. To begin the copying operation, this element is slowly moved away from the reference bit and into a transverse field strong enough to abolish its bistability. This manipulation serves to smoothly and continuously change the element from its initial zero state to a unistable state. The element is then gradually moved out of the transverse field toward the data bit. As the movable bit leaves the region of strong transverse field, its soft mode is biased by the weak longitudinal field from the data bit, thereby making it choose the same direction of magnetization as the data bit. [The region of strong transverse field is assumed to be wide enough that by the time the movable bit reaches its bottom edge, the longitudinal bias field is

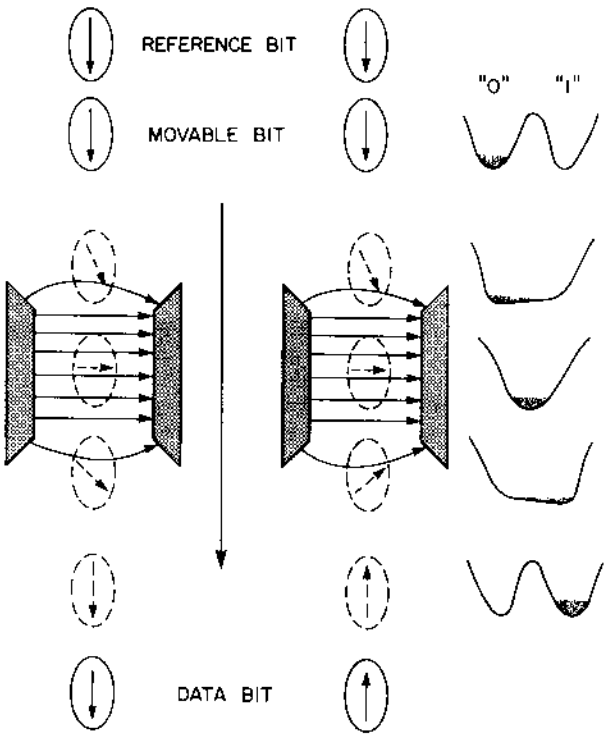

Fig. 14. Reversible copying using a one-domain ferromagnet. The movable bit, initially zero, is mapped into the same state as the data bit (zero in left column; one in center column). Right column shows how the probability density of the movable bit's magnetization, initially concentrated in the "**0**" minimum, is deformed continuously until it occupies the "**1**" minimum, in agreement with a "1" data bit.

due almost entirely to the data bit, with only a negligible perturbation (small compared to kT) from the more distant reference bit.] The overall effect has been to smoothly change the movable element's magnetization direction from agreeing with the reference bit at the top (i.e., zero) to agreeing with the data bit on the bottom. Throughout the manipulation, the movable element's magnetization remains a continuous, single-valued function of its position, and the forces exerted by the various fields on the movable element during the second half of the manipulation are equal and opposite to those exerted during the first half, except for the negligible long-range perturbation of the bias field mentioned earlier, and a viscous damping force (reflecting the finite relaxation time of spontaneous magnetization fluctuations) proportional to the speed of motion. The copying operation can therefore be performed with arbitrarily small dissipation. If carried out in reverse, the manipulation would serve to reversibly erase one of two bits known to be identical, which is the logical inverse of copying onto blank tape.

Like a copying enzyme, this apparatus is susceptible to degradation of its stored information by thermal fluctuations and tunneling. These phenomena, together with the damping coefficient, determine the minimum error probability of which any given apparatus of this type, operating at a given temperature, is capable, and determine a minimum dissipation per step required to operate the apparatus with approximately this error probability. However, as with copying enzymes, there is no fundamental thermodynamic obstacle to making the error probability η and the dissipation ϵ/kT both arbitrarily small, and no practical obstacle to making them both much less than unity.

Bistable magnetic elements of this sort could also, in principle, be used to perform the reversible measurement required by Maxwell's demon. In Figure 15 a diamagnetic particle trapped in the right side of a Maxwell's demon apparatus introduces a weak upward bias in the mode-softening horizontal magnetic field, which yields an upward magnetization of the bistable element when the horizontal field is gradually turned off.

What would happen if a similar manipulation were used to perform a logically *irreversible* operation, e.g., restoring a bistable element that might initially be in either of two states to a standard state? An apparatus for doing this is shown in Figure 16: a memory element which might be magnetized either up or down is moved gradually into a transverse field, rotated slightly counterclockwise to bias the magnetization downward, moved out of the field, and rotated back again, leaving it in the down or zero state.

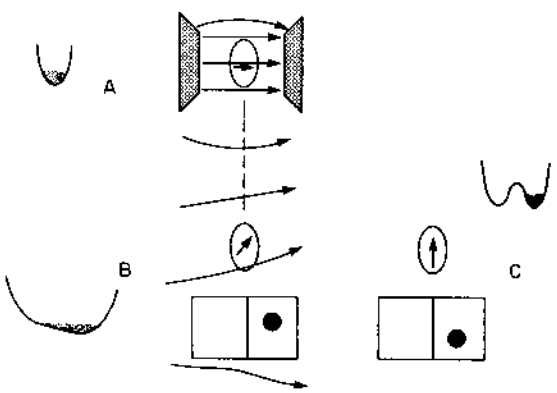

Fig. 15. A one-domain ferromagnet used to make measurements for Maxwell's demon: The bistable element has been moved out of strong transverse field (A) just far enough to create a soft mode (B), which is biased by the perturbation of the field by a diamagnetic Brownian particle in the right side of a Maxwell's demon apparatus. Slowly turning the field off (C) completes the measurement. Peripheral diagrams show potential energy and probability density of the ferromagnetic element's magnetization direction.

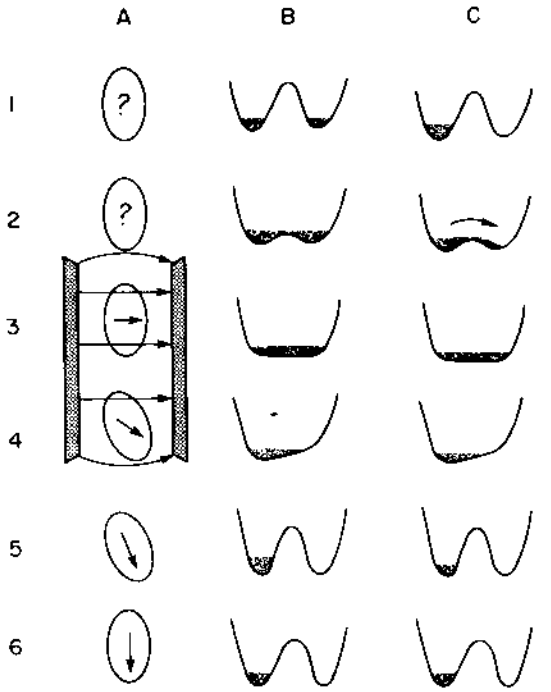

Fig. 16. Erasure of a bistable one-domain ferromagnet. Column A: the bistable element, which may be magnetized either up or down (1), is moved gradually (2) into a transverse field that abolishes its bistability symmetrically (3). It is then rotated slightly counterclockwise (4), to bias the soft mode downward, removed from the field (5), and rotated back again, leaving it in the down or zero state (6). Column B: Evolution of the probability density when the manipulation just described is used to erase a random, unknown bit. Column C: Behavior of the probability density when the manipulation is applied to a known bit (here zero). An irreversible entropy increase of $k\ln 2$ occurs at stage 2, when the probability density leaks out of the initially occupied minimum.

Although such many-to-one mappings have been called logically irreversible, in a subtler sense they may be reversible or not depending on the data to which they are applied. If the initial state in Figure 16 is truly unknown, and properly describable by a probability equidistributed between the two minima (Figure 16B), then undoing the manipulation of Figure 16 exactly restores the initial state of ignorance, and the operation can be viewed as reversible. It is also thermodynamically reversible: the $kT\ln 2$ of work required to carry out the manipulation is compensated by a decrease of $k\ln 2$ in the bistable element's entropy. This expenditure of external work to decrease a system's entropy is analogous to the isothermal compression of a gas to half its volume.

If, on the other hand, the initial state of the memory element were known (e.g., by virtue of its having been set during some intermediate stage of computation with known initial data) the operation would be irreversible:

undoing it would not restore the initial state, and the conversion of $kT \ln 2$ of work into heat in the surroundings would not be compensated by a decrease in the bistable element's entropy. The irreversible entropy increase occurs at the point indicated by the arrow in Figure 16C, when the system's probability density leaks from the minimum it was originally in to fill both minima (as it could have done all along had the initial data been unknown). This is analogous to the free expansion of gas into a previously evacuated container, in which the gas increases its own entropy without doing any work on its environment.

The normal biochemical mechanism by which RNA is destroyed when it is no longer needed (Watson, 1970) provides another example of logical irreversibility. As indicated before, the synthesis of RNA by RNA polymerase is a logically reversible copying operation, and under appropriate (nonphysiological) conditions, it could be carried out at an energy cost of less than kT per nucleotide. The thermodynamically efficient way to get rid of an RNA molecule would be to reverse this process, i.e., to take the RNA back to the DNA from which it was made, and use an enzyme such as RNA polymerase with a slight excess of pyrophosphate to perform a sequence-specific degradation, checking each RNA nucleotide against the corresponding DNA nucleotide before splitting it off. This process does not occur in nature; instead RNA is degraded in a nonspecific and logically irreversible manner by other enzymes, such as polynucleotide phosphorylase. This enzyme catalyzes a reversible reaction between an RNA strand and free phosphate (maintained at high concentration) to split off successive nucleotides of the RNA as nucleotide phosphate monomers. Because the enzyme functions in the absence of a complementary DNA strand, the removal of each nucleotide is logically irreversible, and when running backwards (in the direction of RNA synthesis) the enzyme is about as likely to insert any of the three incorrect nucelotides as it is to reinsert the correct one. This logical irreversibility means that a fourfold higher phosphate concentration is needed to drive the reaction forward than would be required by a sequence-specific degradation. The excess phosphate keeps the enzyme from running backwards and synthesizing random RNA, but it also means that the cycle of specific synthesis followed by nonspecific degradation must waste about $kT \ln 4$ per nucleotide even in the limit of zero speed. For an organism that has already spent around $20kT$ per nucleotide to produce the RNA with near maximal speed and accuracy, the extra $1.4kT$ is obviously a small price to pay for being able to dispose of the RNA in a summary manner, without taking it back to its birthplace.

Corollary to the principle that erasing a random tape entails an entropy increase in the environment is the fact that a physical system of low entropy (e.g., a gas at high pressure, a collection of magnetic domains or individual

atomic spins, all pointing "down," or more symbolically, a tape full of zeros) can act as a "fuel," doing useful thermodynamic or mechanical work as it randomizes itself. For example, a compressed gas expanding isothermally against a piston increases its own entropy, meanwhile converting waste heat from its surroundings into an equivalent amount of mechanical work. In an adiabatic demagnetization experiment, the randomization of a set of aligned spins allows them to pump heat from a cold body into warmer surroundings. A tape containing N zeros would similarly serve as a fuel to do $NkT\ln 2$ of useful work as it randomized itself.

What about a tape containing a computable pseudorandom sequence like the first N digits of the binary expansion of pi? Although the pattern of digits in pi looks random to begin with, and would not accomplish any useful work if allowed to truly randomize itself in an adiabatic demagnetization apparatus of the usual sort, it could be exploited by a slightly more complicated apparatus. Since the mapping (N zeros)↔(first N bits of pi) is one-to-one, it is possible in principle to construct a reversible computer that would execute this mapping physically, in either direction, at an arbitrarily small thermodynamic cost per digit. Thus, given a pi tape, we could convert it into a tape full of zeros, then use that as fuel by allowing it to truly randomize itself at the expense of waste heat in the surroundings.

What about a tape containing N bits generated by coin tossing? Of course coin tossing might accidentally yield a tape of N consecutive zeros, which could be used as fuel; however, the typical result would be a sequence with no such unusual features. Since the tape, after all, contains a specific sequence, $NkT\ln 2$ of work ought to be made available when it is mapped, in a one-to-many manner, onto a uniform distribution over all 2^N N-bit sequences. However, because there is no concise description of the way in which this specific sequence differs from the bulk of other sequences, no simple apparatus could extract this work. A complicated apparatus, containing a verbose description or verbatim copy of the specific sequence, could extract the work, but in doing so it would be doing no more than canceling the specific random sequence against the apparatus's copy of it to produce a blank tape, then using that as fuel. Of course there is a concise procedure for converting a random tape into a blank tape: erase it. But this procedure is not logically reversible, and so would require an input of energy equal to the fuel value of the blank tape it produced.

Finally, suppose we had seven identical copies of a typical random tape. Six of these could be converted into blank tapes by a logically and thermodynamically reversible process, e.g., subtracting one copy from another, digit by digit, to produce a tape full of zeros. The last copy could not be so canceled, because there would be nothing left to cancel it against. Thus the seven identical random tapes are interconvertible to six blank

tapes and one random tape, and would have a fuel value equal to that of six blank tapes. The result of their exploitation as fuel, of course, would be seven *different* random tapes.

6. ALGORITHMIC ENTROPY AND THERMODYNAMICS

The above ideas may be generalized and expressed more precisely using algorithmic information theory, described in an introductory article by Chaitin (1975a), and review articles by Zvonkin and Levin (1970), and Chaitin (1977).

Ordinary information theory offers no solid grounds for calling one N-bit string "more random" than another, since all are equally likely to be produced by a random process such as coin tossing. Thus, in particular, it cannot distinguish bit strings with fuel value from those without. This inability reflects the fact that entropy is an inherently statistical concept, applicable to ensembles but not to the individual events comprising them. In equilibrium statistical mechanics, this fact manifests itself as the well-known impossibility of expressing macroscopic entropy as the ensemble average of some microscopic variable, the way temperature can be expressed as the average of $mv^2/2$: since the entropy of a distribution $\mathbf{p}$ is defined as

$$S[\mathbf{p}] = \sum_x p(x) \log[1/p(x)]$$

obviously there can be no one function $f(x)$, such that for all $\mathbf{p}$,

$$S[\mathbf{p}] = \sum_x p(x) f(x)$$

The absence of a microscopic quantity corresponding to entropy is a nuisance both practically and conceptually, requiring that entropy always be measured by indirect calorimetric methods, both in the laboratory and in Monte Carlo and molecular-dynamics "computer experiments" (Bennett, 1975), and frustrating the natural desire of molecular model builders to regard an individual molecular configuration as having an entropy.

Though it does not help the practical problem, the notion of algorithmic entropy resolves the conceptual problem by providing a microstate function, $H(x)$, whose average is very nearly equal to the macroscopic entropy $S[\mathbf{p}]$, not for all distributions $\mathbf{p}$ (which would be impossible) but rather for a large class of distributions including most of those relevant to statistical mechanics. Algorithmic entropy is a measure, not of any obvious physical property of the microstate x, but rather of the number of bits

required to describe x in an absolute mathematical sense, as the output of a universal computer. Algorithmic entropy is small for sequences such as the first million digits of pi, which can be computed from a small description, but large for typical sequences produced by coin tossing, which have no concise description. Sequences with large algorithmic entropy cannot be erased except by an irreversible process; conversely, those with small algorithmic entropy can do thermodynamic work as they randomize themselves.

Several slightly different definitions of algorithmic entropy have been proposed; for definiteness we adopt the definition of Levin (Gacs, 1974; Levin, 1976) and Chaitin (1975b) in terms of self-delimiting program size: the algorithmic entropy $H(x)$ of a binary string x is the number of bits in the smallest self-delimiting program causing a standard computer to embark on a halting computation with x as its output. (A program is self-delimiting if it needs no special symbol, other than the digits 0 and 1 of which it is composed, to mark its end.) The algorithmic entropy of discrete objects other than binary strings, e.g., integers, or coarse-grained cells in a continuous phase space, may be defined by indexing them by binary strings in a standard way. A string is called "algorithmically random" if it is not expressible as the output of a program much shorter than the string itself. A simple counting argument shows that, for any length N, most N-bit strings are algorithmically random (e.g., there are only enough $N-10$ bit programs to describe at most $1/1024$ of all the N-bit strings).

At first it might appear that the definition of H is extremely machine dependent. However, it is well known that there exist computers which are universal in the strong sense of being able to simulate any other computer with at most an additive constant increase in program size. The algorithmic entropy functions defined by any two such machines therefore differ by at most $O(1)$, and algorithmic entropy, like classical thermodynamic entropy, may be regarded as well defined up to an additive constant.

A further noteworthy fact about $H(x)$ is that it is not an effectively computable function of x: there is no uniform procedure for computing $H(x)$ given x. Although this means that $H(x)$ cannot be routinely evaluated the way $x!$ and $\sin(x)$ can, it is not a severe limitation in the present context, where we wish chiefly to prove theorems about the relation between H and other entropy functions. From a more practical viewpoint, the molecular model builder, formerly told that the question "what is its entropy?" was meaningless, is now told that the question is meaningful but its answer cannot generally be determined by looking at the model.

It is easy to see that simply describable deterministic transformations cannot increase a system's algorithmic entropy very much (i.e., by more than the number of bits required to describe the transformation). This

follows because the final state can always be described indirectly by describing the initial state and the transformation. Therefore, for a system to increase its algorithmic entropy, it must behave probabilistically, increasing its statistical entropy at the same time. By the same token, simply describable reversible transformations (1 : 1 mappings) leave algorithmic entropy approximately unchanged.

Algorithmic entropy is a microscopic analog of ordinary statistical entropy in the following sense: if a macrostate **p** is *concisely describable*, e.g., if it is determined by equations of motion and boundary conditions describable in a small number of bits, then its statistical entropy is nearly equal to the ensemble average of the microstates' algorithmic entropy. In more detail, the relation between algorithmic and statistical entropy of a macrostate is as follows:

$$S_2[\mathbf{p}] < \sum_x p(x)H(x) \leqslant S_2[\mathbf{p}] + H(\mathbf{p}) + O(1) \tag{1}$$

Here $S_2[\mathbf{p}] = \Sigma_x p(x)\log_2[1/p(x)]$, the macrostate's statistical entropy in binary units, measures the extent to which the distribution **p** is spread out over many microstates; $H(x)$, the microstate's algorithmic entropy, measures the extent to which a particular microstate x is not concisely describable; and $H(\mathbf{p})$, the algorithmic entropy of **p**, is the number of bits required to describe the distribution **p**.

We need to say in more detail what it means to describe a distribution. Strictly construed, a description of **p** would be a program to compute a tabulation of the components of the vector **p**, to arbitrarily great precision in case any of the components were irrational numbers. In fact, equation (1) remains true for a much weaker kind of description: a Monte Carlo program for sampling some distribution **q** not too different from **p**. In this context, a Monte Carlo program means a fixed program, which, when given to the machine on its input tape, causes the machine to ask from time to time for additional bits of input; and if these are furnished probabilistically by tossing a fair coin, the machine eventually halts, yielding an output distributed according to the distribution **q**. For **q** not to be too different from **p** means that $\Sigma_x p(x)\log_2[p(x)/q(x)]$ is of order unity.

For typical macrostates considered in statistical mechanics, the equality between statistical entropy and the ensemble average of algorithmic entropy holds with negligible error, since the statistical entropy is typically of order 10^{23} bits, while the error term $H(\mathbf{p})$ is typically only the few thousand bits required to describe the macrostate's determining equations of motion, boundary conditions, etc. Macrostates occurring in nature (e.g., a gas in a

box with irregular walls) may not be so concisely describable, but the traditional approach of statistical mechanics has been to approximate nature by simple models.

We now sketch the proof of equation (1). The left inequality follows from the convexity of the log function and the fact that, when algorithmic entropy is defined by self-delimiting programs, $\Sigma_x 2^{-H(x)}$ is less than 1. The right inequality is obtained from the following inequality, which holds for all x:

$$H(x) \leqslant H(\mathbf{q}) + \log_2[1/q(x)] + O(1) \tag{2}$$

Here $\mathbf{q}$ is a distribution approximating $\mathbf{p}$ that can be exactly sampled by a Monte Carlo program of size $H(\mathbf{q})$. The additive $O(1)$ constant again depends on the universal computer but not on $\mathbf{q}$ or x. Equation (2) follows from the fact that one way of algorithmically describing x is to first describe the distribution $\mathbf{q}$ and then describe how to locate x within this distribution. The proof that a self-delimiting program of size $\log_2[1/q(x)] + O(1)$ bits suffices to compute x, given a Monte Carlo routine for sampling $\mathbf{q}$, is given by Chaitin (1975b, Theorem 3.2); slightly weaker versions of the idea are easy to understand intuitively. Equation (1) is finally obtained by summing equation (2) over the distribution $\mathbf{p}$, and applying the criterion of closeness of approximation of $\mathbf{p}$ by $\mathbf{q}$.

In conjunction with the second law, equation (1) implies that when a physical system increases its algorithmic entropy by N bits (which it can only do by behaving probabilistically), it has the capacity to convert about $NkT \ln 2$ of waste heat into useful work in its surroundings. Conversely, the conversion of about $NkT \ln 2$ of work into heat in the surroundings is necessary to decrease a system's algorithmic entropy by N bits. These statements are classical truisms when entropy is interpreted statistically, as a property of ensembles. The novelty here is in using algorithmic entropy, a property of microstates. No property of the ensembles need be assumed, beyond that they be concisely describable.

ACKNOWLEDGMENTS

The work on enzymatic and clockwork Turing machines was done in 1970–72, at Argonne National Laboratory (Solid State Science Division), under the auspices of the U.S. Atomic Energy Commission. I wish to thank Rolf Landauer and Gregory Chaitin for years of stimulating discussions of reversibility and entropy, Michael Chamberlain for background information on polymerases, and John Slonczewski for background information on ferromagnets.

REFERENCES

Benioff, Paul (1982) to appear in *Journal of Statistical Mechanics.*

Bennett, C. H. (1973). "Logical Reversibility of Computation", *IBM Journal of Research and Development*, **17**, 525–532.

Bennett, C. H. (1975). "Efficient Estimation of Free Energy Differences from Monte Carlo Data," *Journal of Computational Physics*, **22**, 245–268.

Bennett, C. H. (1979). "Dissipation-Error Tradeoff in Proofreading," *BioSystems*, **11**, 85–90.

Chaitin, G. (1975a). "Randomness and Mathematical Proof," *Scientific American*, **232**, No. 5, 46–52.

Chaitin, G. (1975b). "A Theory of Program Size Formally Identical to Information Theory," *Journal of the Association for Computing Machinery*, **22**, 329–340.

Chaitin, G. (1977). "Algorithmic Information Theory," *IBM Journal of Research and Development*, **21**, 350–359, 496.

Brillouin, L. (1956). *Science and Information Theory* (2nd edition, 1962), pp. 261–264, 194–196. Academic Press, London.

Fredkin, Edward, and Toffoli, Tommaso, (1982). "Conservative Logic," MIT Report MIT/LCS/TM-197; *International Journal of Theoretical Physics*, **21**, 219.

Gacs, P. (1974). "On the Symmetry of Algorithmic Information," *Soviet Mathematics Doklady*, **15**, 1477.

Hopfield, J. J. (1974). *Proceedings of the National Academy of Science USA*, **71**, 4135–4139.

Keyes, R. W., and Landuer, R. (1970). *IBM Journal of Research and Development*, **14**, 152.

Landauer, R. (1961). "Irreversibility and Heat Generation in the Computing Process," *IBM Journal of Research and Development*, **3**, 183–191.

Levin, L. A. (1976). "Various Measures of Complexity for Finite Objects (Axiomatic Description)," *Soviet Mathematics Doklady*, **17**, 522–526.

Likharev, K. (1982). "Classical and Quantum Limitations on Energy Consumption in Computation," *International Journal of Theoretical Physics*, **21**, 311.

McCarthy, John (1956). "The Inversion of Functions Defined by Turing Machines," in *Automata Studies*, C. E. Shannon and J. McCarthy, eds. Princeton Univ. Press, New Jersey.

Ninio, J. (1975). *Biochimie*, **57**, 587–595.

Reif, John H. (1979). "Complexity of the Mover's Problem and Generalizations," Proc. 20'th IEEE Symp. Found. Comp. Sci., San Juan, Puerto Rico, pp. 421–427.

Szilard, L. (1929). *Zeitschrift für Physik*, **53**, 840–856.

Toffoli, Tommaso (1980). "Reversible Computing," MIT Report MIT/LCS/TM-151.

Toffoli, Tommaso (1981). "Bicontinuous Extensions of Invertible Combinatorial Functions," *Mathematical and Systems Theory*, **14**, 13–23.

von Neumann, J. (1966). Fourth University of Illinois lecture, in *Theory of Self-Reproducing Automata*, A. W. Burks, ed., p. 66. Univ. of Illinois Press, Urbana.

Watson, J. D. (1970). *Molecular Biology of the Gene* (2nd edition). W. A. Benjamin, New York.

Zvonkin, A. K., and Levin, L. A. (1970). "The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by Means of the Theory of Algorithms," *Russian Mathematical Surveys*, **25**, 83–124.