Molecular Computing Machines

Yaakov Benenson
Ehud Shapiro
Weizmann Institute of Science, Rehovot, Israel

INTRODUCTION

Biopolymers such as nucleic acids and proteins encode biological data and may be viewed as strings of chemical letters. While electronic computers manipulate strings of 0's and 1's encoded in electric signals, biologically encoded data might, in principle, be manipulated by biochemical means. During the last decade, several approaches to compute with biomolecules were developed, and the field has become known as biomolecular or DNA computing. The approaches varied widely with respect to the model of computation they employed, the problems they attempted to solve, and the degree of human intervention. One approach focused on the application of the Turing machine model and, more generally, string-processing automata to biomolecular information processing. Its goal is to construct computers made of biomolecules that are capable of autonomous conversion of an input data-encoding molecule to an output molecule according to a set of rules defined by a molecular program. Here we survey the field of biomolecular computing machines and discuss possible future directions.

BACKGROUND

The seminal work of Adleman[1] demonstrated that commonly used biochemical manipulations of DNA can be utilized to solve real-world computational problems and initiated the field of biomolecular computing. In the biomolecular approach to computing, the computational paradigm is chosen to fit the capabilities of biomolecules, rather than adapting the biomolecular machinery to computational schemes borrowed from electronic computers.[2] The problems initially solved by DNA computing were so-called ''combinatorial problems.'' An example of such a problem is the traveling salesman problem, which is to find the most efficient route through several cities given a distances chart between them, passing through each city exactly once. Solving the problem can be performed by calculating all possible routes that pass exactly once through each city, comparing them and choosing the shortest one. As the number of potential routes is exponential in the size of the problem, this computation may require an exponential number of steps. More efficient solution methods are not known for the traveling salesman problem and for similar such problems, termed NP-hard. It was hoped that the potential massive parallelism of DNA manipulation could speed up the solution of NP-hard problems. The DNA computing technique employed to solve the traveling salesman problem included 1) generation of all possible solution candidates (e.g., various routes) encoded in DNA strands and selection of the correct ones, 2) their amplification and detection by known molecular biology techniques, and 3) isolation and characterization of the shortest one. Computational problems solved in vitro with variations of this approach encompassed instances of Hamiltonian Path,[1] SAT,[3] maximal clique,[4] and ''knight move''[5] problems. The computer, i.e., the physical system that produced a solution, comprised the biomolecules themselves, the laboratory equipment required to realize their biochemical manipulation, and the laboratory personnel who operated the laboratory equipment, performing the operations required to execute the computation. Therefore while these computing systems used biomolecules for computation, they realized laboratory-scale, rather than molecular-scale, computers.

A second direction in DNA computing, proposed by Winfree,[6,7] uses self-assembly of DNA tiles.[8,9] It relies on the mathematical theory of tiling. One result of this theory discovered by Wang[10] is that aperiodic assembly of appropriately designed tiles emulates the operation of a Turing machine, a universal computer. The tiles have colored edges, and they may be assembled once two adjacent tiles have edges of the same color. DNA tiles[8] are relatively rigid flat constructs with four sticky ends, with one sticky end emulating one edge of a tile and different sticky ends emulating different colors. DNA tiles make contact through complementary sticky ends, emulating recognition by the same color. Initial breakthrough in this area was achieved by constructing a periodic two-dimensional crystal from DNA tiles, based on Wang assembly rules. The first actual computation performed by this technique was a cumulative XOR (exclusive OR) logical operation on a string of four binary bits.[9] In this experiment, an input string was built from alphabet tiles (either ''0'' or ''1'') and then a second row of tiles

DOI: 10.1081/E-ENN 120013375

self-assembled upon it. The first tile of the second row contained the result of the XOR operation between the first two bits, and each subsequent tile performed the operation between the intermediate cumulative result and the next unprocessed bit. Besides the potential to realize universal computation through Turing machine emulation, the technique of DNA tiles self-assembly may become a basis for fabrication of smart, aperiodic materials on a nanoscale, as suggested by several recent results.[11,12]

A third direction in DNA computing is an attempt to realize the vision,[13] recalled by Adleman in the conclusion to his seminal paper of a programmable, autonomous, molecular-scale computer: ''In the future, research in molecular biology may provide improved techniques for manipulating macromolecules. Research in chemistry may allow for the development of synthetic designer enzymes. One can imagine the eventual emergence of a general purpose computer consisting of nothing more than a single macromolecule conjugated to a ribosomelike collection of enzymes that act on it.'' This paradigm of biomolecular computers is the focus of our review. It views a DNA strand as a string or a tape that functions as the input as well as the memory storage for *automata* such as a finite automaton or a Turing machine.[13] This paradigm is inspired by the realization that some biomolecular machines in the living cell are essentially simple automata operating on digital information encoded in directional biopolymers.[14,15] An automaton operates by scanning a tape of symbols one symbol at a time, possibly modifies one symbol in each step, moving to an adjacent symbol and changing its state according to a predefined set of the transition rules. The tape of symbols may be naturally encoded in a polar biopolymer such as DNA or RNA. The transition rules of the machine may be encoded by transition molecules similar to tRNA. A transition, i.e., the physical modification of the input according to the transition rules, may be accomplished, in principle, by a combination of different processing enzymes. Taking this viewpoint, DNA and RNA polymerases, the ribosome, and recombinases can all be viewed as simple molecular automata. For example, RNA polymerase is, mathematically speaking, a so-called finite state transducer, which translates a string over the alphabet {A, T, C, G} into a string over the alphabet {A, U, C, G} according to a simple translation table. An artificial molecular automaton may be able to operate autonomously, realizing a truly molecular-scale computer. Such a computer could have several important applications, discussed below.

The concept of a biomolecular computer was first introduced by Bennett[13] in 1982 as a hypothetical design for an energy-efficient computer. In this conceptual design, a set of artificial enzymes encoded the transition table of the machine and operated on RNA-based data tape. The design did not include any concrete implementation details. Several detailed designs were proposed since then. Rothemund[16] and Smith[17] proposed models for molecular implementations of Turing machines. Garzon et al.[18] designed a model of finite automata and Sakamoto et al.[19,20] implemented a semiautonomous state machine that could perform state transitions.

MOLECULAR AUTOMATA

Automata

Generally, an automaton consists of 1) a data tape divided into cells, each containing a symbol selected from the tape alphabet, and 2) a finite-state device driven by transition rules. The device is positioned over one of the cells and is in one of a finite number of internal states. Depending on the symbol read and the internal state, a transition rule instructs the device to write a new symbol, change state, and move one cell to the left or to the right. The Turing machine[21] is the most general automaton, capable of writing on the tape as well as moving in either direction. Fig. 1A demonstrates a Turing machine with two symbols and two states, with the upper part of the panel showing the application of one transition rule. Each rule is of the form *initial state*, *current symbol*→*new state*, *new symbol*, *direction of movement* (R=right, L=left). A more restricted, yet important, class of automata is called finite-state acceptors (finite automata for short).[22] A finite automaton is a unidirectional read-only Turing machine. Its input is a finite string of symbols. It is initially positioned on the leftmost input symbol in a default initial state and, in each transition, moves one symbol to the right, possibly changing its internal state (Fig. 1B is an example of a computation step of a finite automaton). Each of its transition rules specifies a next state based on the current state and current symbol. The computation terminates after the last input symbol is processed. Alternatively, it may suspend without completion when no transition rule applies. Some states are deemed accepting and the automaton accepts an input if there is a computation with this input that ends in an accepting final state. Otherwise, it is said to reject the input.

A finite automaton with two states and an alphabet of two symbols is shown in Fig. 1C. It determines whether a string of symbols over the alphabet {a, b} contains an even number of a's. On a diagram, an incoming arrow represents an initial state, and a double circle represents an accepting state. Below the diagram, a sample computation over an input *abba* shows the intermediate configurations obtained during the sequential application of the transition rules.

A two-state, two-symbol automaton can have eight possible transition rules. The programming of such an

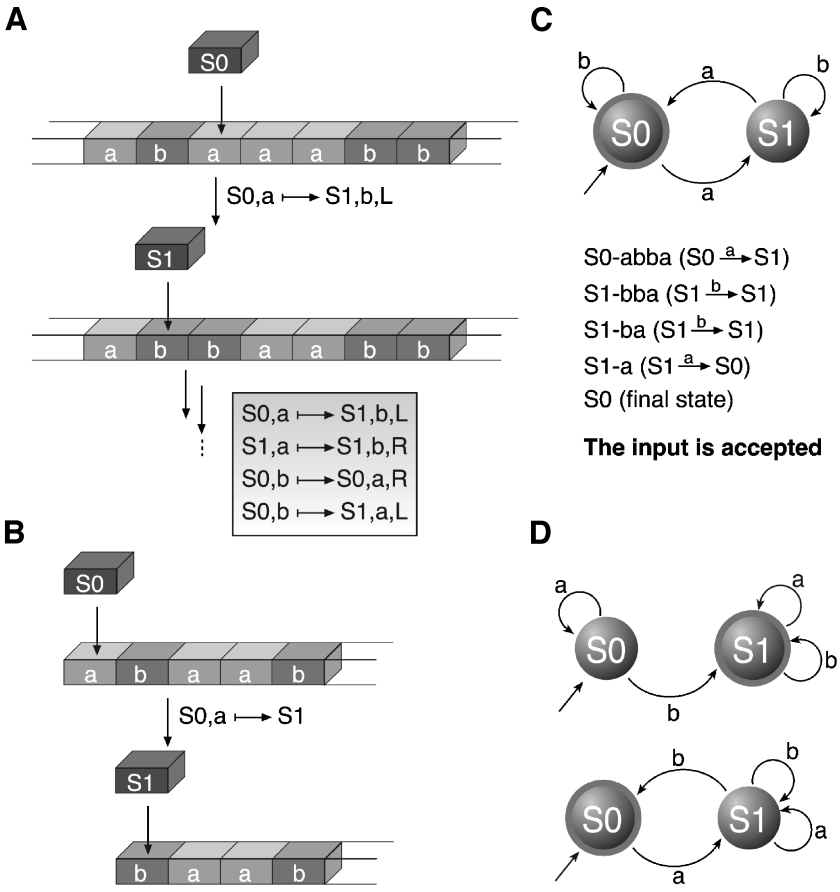

Fig. 1 Examples of automata.

automaton amounts to selecting the transition rules and determining the accepting states. Fig. 1D shows some examples of additional final automata. The topmost automaton determines whether an input string contains at least one *b* symbol. A second one determines if an input string begins with *a* and ends with *b*. It is an example of a nondeterministic automaton with two transitions (S1, $b \rightarrow$S0 and S1, $b \rightarrow$S1) applicable to the same configuration. A computation ending in an accepting state uses S1, $b \rightarrow$S1 for all *b* symbols except the last, and uses S1, $b \rightarrow$S0 for the last *b*.

Early Designs of Molecular Automata

Bennett[13] described a ''truly chemical Turing machine'' with a linear tape analogous to RNA, where the internal state and head location are realized by a special chemical modifier attached to one of the nucleotides. Each transition rule is realized by a hypothetical ''enzyme'' that exclusively recognizes a unique combination of a nucleotide and its modifier, replaces a nucleotide by an output symbol, and attaches a next-state modifier to one of the adjacent nucleotides, according to the desired head movement (Fig. 2).

In Fig. 2, a transition molecule **7** that recognizes a combination of the symbol *a* and the state S0 loads itself with the molecule for the symbol *b* and a molecule for the state S1. The loaded molecule **2** reversibly attaches itself to a data tape **1**. An intermediate complex **3** forms through new chemical bonds between the transition molecule and the symbol *a* and state S0, between the new symbol *b* and the data tape, and between the adjacent symbol *b* and the new state S1 (dotted lines). In the next intermediate **4**, the old symbol *a* and state S0 become attached to the transition molecule and are detached from the data tape; the new symbol *b* is inserted into the data tape; the new state S1 attaches to the symbol *b* that lies to the right of the site of newly inserted symbol. The transition molecule **6** dissociates from the completely modified data tape **5** and is subsequently stripped of the attached old state and symbol.

Bennett introduced several logical elements that remain relevant till this day. First, he proposed to encode a data tape in a single biopolymer, using a natural

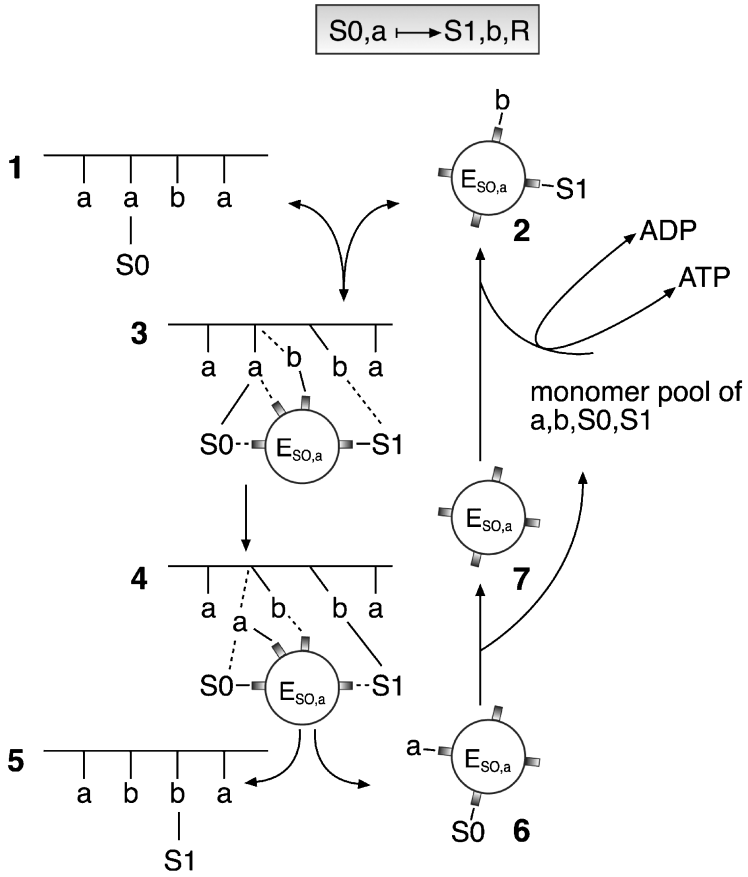

Fig. 2 An example of a single transition performed by Bennett's hypothetical chemical Turing machine. The rule implemented is S0, $a \rightarrow$ S1, *b*, R. (From Ref. [13] © Kluwer Academic/Plenum Publishers.)

''alphabet.'' Second, he introduced the important concept of a ''transition molecule,'' i.e., representation of each transition rule by a separate molecule or molecular assembly.

Following work on molecular automata dealt with realizing this concept. Different ways to encode tape symbols and machine states and to build transition molecules were proposed and various biochemical transition mechanisms were considered.

Rothemund[16] proposed a detailed design for a molecular Turing machine that utilized a common DNA structural motif known as a ''cohesive terminus'' or a ''sticky end.'' A sticky end is a short (one to six nucleotides) stretch of single-stranded DNA emerging from the double-stranded DNA molecule of a potentially unlimited length. The advantage of a sticky end as compared with the dsDNA is that it is reactive compared with dsDNA. Molecules with sticky ends may interact once the DNA sequences of their sticky ends are complementary, irrespective to the sequence of their double-stranded part. This technique is extensively used in recombinant DNA technology. Rothemund's hypothetical computer comprised a data tape and transition molecules made of DNA and hardware containing DNA ligase and restriction enzymes. Ligase is an enzyme that may glue together fragments of DNA that have complementary sticky ends. Restriction enzymes recognize specific locations in the double-stranded DNA and cut inside of near this location, forming two fragments with complementary sticky ends. Rothemund pioneered an encoding system of ''frame shifts,'' where a long stretch of double-stranded DNA encoded a symbol while shorter sticky ends derived from this stretch encoded state-symbol combinations. This entailed a particular design of the transition molecules, using SII-type restriction enzymes that cut DNA outside their recognition sequence. The machine was not designed to be autonomous as it required a number of manual steps to perform a single transition (Fig. 3).

Fig. 3 describes a single transition S0, a $\rightarrow$ S1, b, R as implemented by Rothemund's hypothetical Turing machine. In this design, each data symbol is represented by a stretch of dsDNA flanked by invariant left (**L**) and right

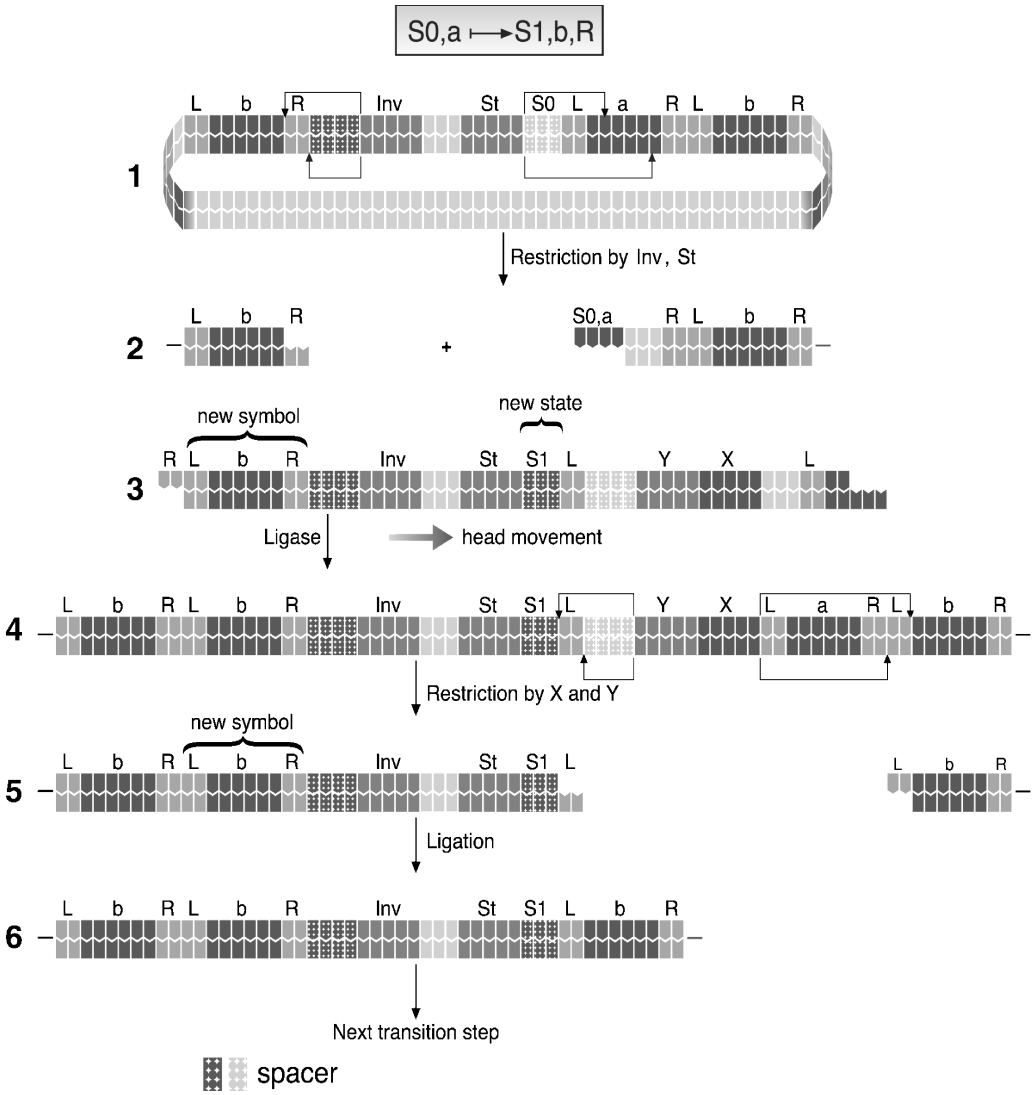

Fig. 3 A single transition S0, $a \rightarrow$ S1, b, R as implemented by the Rothemund's hypothetical Turing machine. (From Ref. [16] © American Mathematical Society.)

(**R**) short sequences. The starting circular double-stranded DNA structure **1** represents a data string *bab* and a Turing machine head, located at a symbol *a*. Inside a ''head,'' there are two SII-type enzyme recognition sites. One is denoted **Inv** and another **St** for ''state enzyme.'' **Inv** enzyme invariantly cuts in the **R** region of a symbol that lies to the left of the head. **St** enzyme cuts inside the current symbol *a*, and the exact restriction site is determined by the length of a spacer between the **St** recognition site and an **L** region of the current symbol. This spacer is denoted as **S0**.

After double restriction by **Inv** and **St**, the enzymes are removed and a modified data tape **2** with two sticky ends is formed. One is inside the **R** region of the leftmost *b* symbol. Another lies within the coding sequence of a current *a* symbol. Because the exact structure of this sticky ends depends on the state spacer length, it contains information on both the current symbol and the current state. This sticky end is recognized by the transition molecule **3**, which encodes a rule S0, $a \rightarrow$ S1, *b*, R. It contains two sticky ends: one recognizes the current state-symbol sticky end of the data tape, and another one recognizes the sticky end within the **R** region of the left-hand symbol. Its dsDNA region contains, from left to right, an encoding for a new symbol *b*, a spacer and a recognition site of the **Inv** enzyme, a recognition site of the **St** enzyme, a new state spacer (S1 in this case), **L** region, another spacer, two recognition sites of the auxiliary SII-type enzymes **X** and **Y**, and another **L** region.

The solution containing the transition molecules is added to the cleaved data tape **2**. Because real computation would require many different kinds of transition molecules, the existence of an invariant sticky end in a data tape would allow it to react nonselectively with different transition molecules. Therefore the original design contained only right-hand state-symbol specific sticky end. The left-hand sticky end was exposed by yet another restriction enzyme after correct ligation to the state-symbol sticky end and washout of the useless transitions (not shown on the figure). Following double ligation and insertion of the transition molecule, an intermediate structure **4** is formed. At this stage, the new symbol *b* is inserted into the tape. The head is also inserted, with the new state encoded by the new state spacer. However, the previous symbol *a* is regenerated. It needs to be excised by means of the enzymes **X** and **Y**. The enzymes form two complementary sticky ends in the intermediate **5**: one in the **L** region of the right-hand symbol *b*, and another one in the **L** region preinserted in the transition molecule. After their ligation, the next legal configuration **6** is formed. The **St** enzyme is now positioned at the correct distance from the next symbol *b*. Thus this multistep transition process results in an insertion of a new symbol *b*, excision of a previous symbol *a*, and change of the machine state from S0 to S1.

Sakamoto et al.[19,20] described a different approach to biomolecular state machines. While their system implemented only a fixed state-to-state transition scheme, which is not dependent on any input, it had the advantage of semiautonomous operation. The system was experimentally verified and shown to perform several transitions (Fig. 4).

Fig. 4A shows a set of transition rules. Panel B depicts the molecular transition table. Each state is encoded by unique sequence of a ssDNA of 20–30 nt long. A transition between S1 and S2 is represented by a concatenation of two sequences, one complementary to S1 and another one—to S2. The ''stop'' segment does not allow DNA polymerase to pass through. Fig. 4C shows the initial configuration of the machine. A transition table is concatenated to the initial state S1. Fig. 4D describes a sample computation. Initial state S1 is annealed to its complementary counterpart in the rule S1 → S2 of the transition table and then extended by DNA polymerase to form a DNA stretch for S2. After denaturation and another annealing, the S2 stretch is annealed to its counterpart in the rule S2 → S3 and extended to S3 stretch. The whole

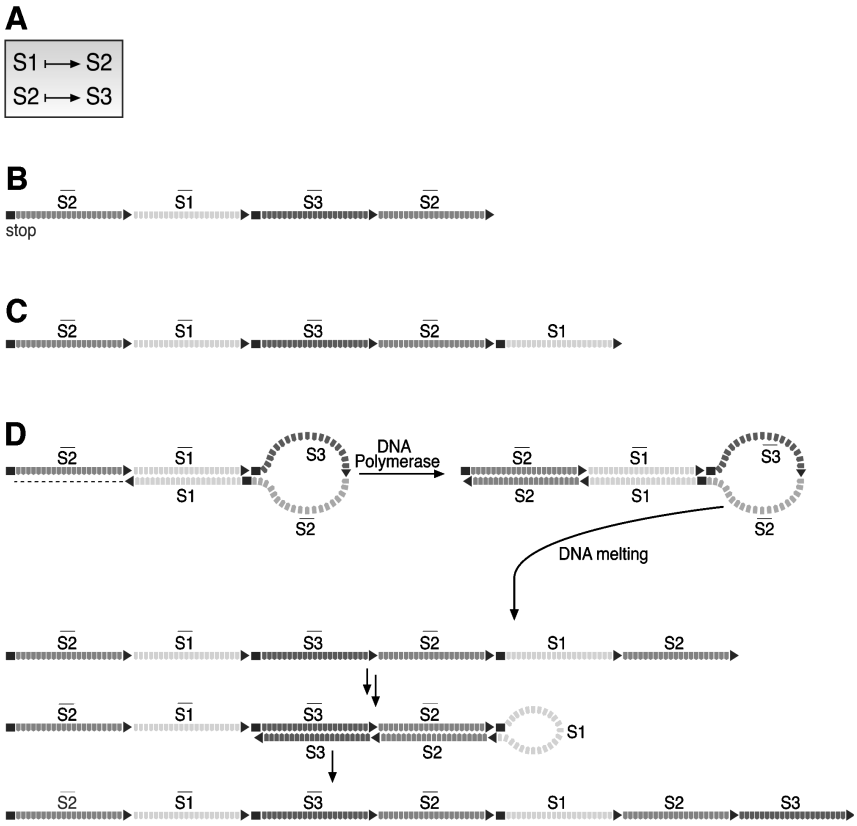

Fig. 4 A molecular state machine of Sakamoto et al.

process is performed in the PCR-like manner, termed ''Whiplash PCR'' with cycles of denaturing, annealing, and polymerase extension.

Shapiro[14,15] proposed a detailed logical design for a molecular Turing machine, with an emphasis on a general-purpose programmable computer that may operate in vivo and interact with its biochemical environment. The design was realized in a working mechanical implementation.

The structural blocks of the design proposed by Shapiro are depicted in Fig. 5. The mechanical computer employs a chain of basic building blocks (Fig. 5A), referred to as *alphabet monomers*, to represent the Turing machine's tape (Fig. 5B), and uses another set of building blocks (Fig. 5C), referred to as *transition molecules*, to encode the machine's transition rules. The computer operates on two polymers simultaneously: the *tape polymer*, representing the Turing machine's tape, and the *trace polymer*, which is a byproduct of the computation constructed incrementally from displaced transition molecules and displaced alphabet monomers and has no analog in the theoretical Turing machine. A transition molecule loaded with an alphabet monomer specifies a computational step of the computer similarly to the way an aminoacyl-tRNA specifies a translation step of the ribosome.[23] The transition encoding is similar to a Wang tile construction[10] which is also at the basis of DNA computing via self-assembly.[6–9] The set of loaded transition molecules constitutes the computer *program* (Fig. 1A). A description of the design and mechanism of operation is shown in Figs. 5 and 6. Fig. 5C shows the formation of a transition molecule and Fig. 5D shows an active transition molecule. An active transition molecule joins the two data polymers. It is embedded in the tape polymer and represents the location of the Turing machine's read/write head as well as the machine's internal state. At the same time, the active transition molecule is the terminal molecule of the trace polymer, representing the most recent transition of the computation. Fig. 5E schematically depicts the computer (hardware). The computer is made of two subunits, referred to as *small* and *large*, each with a tunnel called the *small tunnel* and the *large tunnel*, respectively.

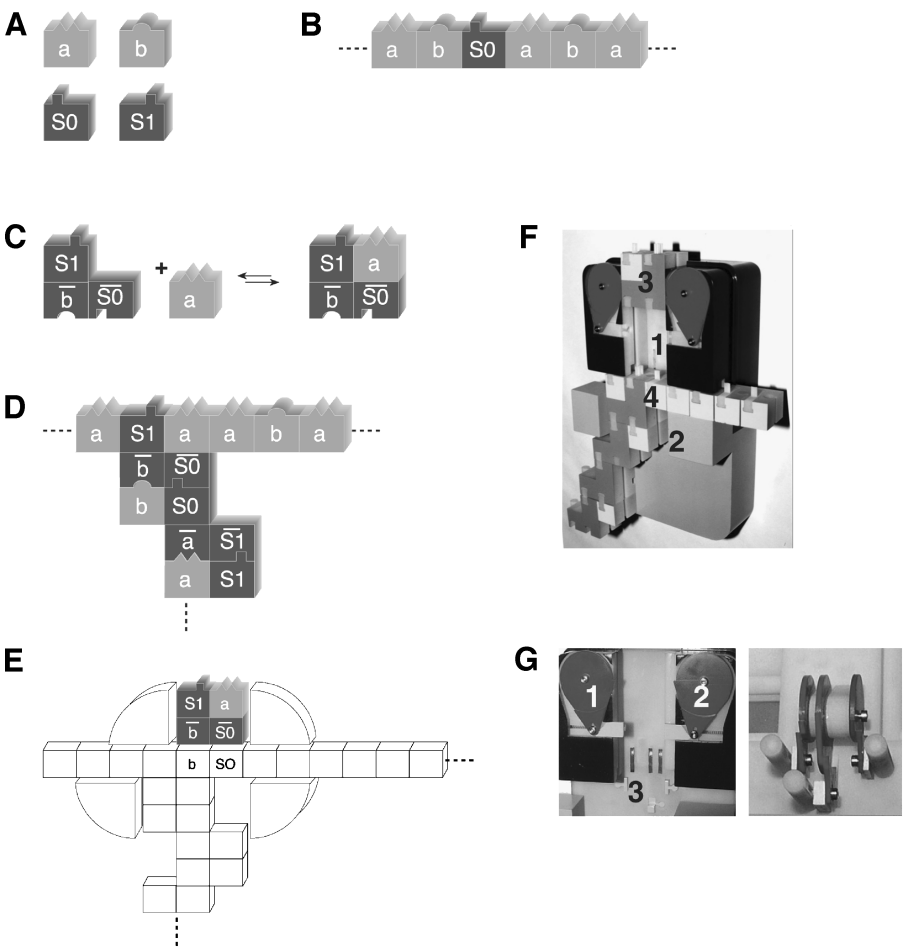

Fig. 5 Structural blocks of the Shapiro's mechanical Turing machine.

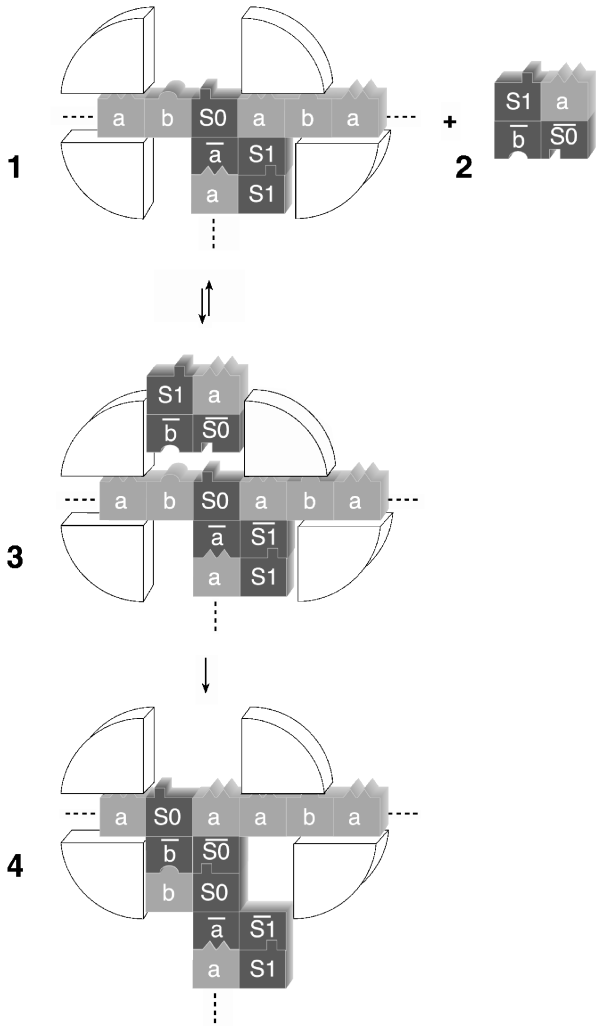

Fig. 6 Operational cycle of the Shapiro's mechanical Turing machine.

The small tunnel provides incoming loaded transition molecules with access to the active transition molecule and to its adjacent alphabet monomer. Access is controlled by gating mechanisms that block transition molecules that are ill-formed or do not match the current state and current tape symbol. These mechanical analogs of allosteric conformational changes open the channel only when a valid incoming transition molecule approaches. The large tunnel holds the active transition molecule and the tail of the trace polymer being constructed.

The actual mechanical computer is $18 \times 29 \times 9$ cm as shown in Fig. 5F. The small tunnel **1** is part of the small subunit and is 2 units wide. The large tunnel **2** is part of the large subunit and is 3 units wide, so that it can accommodate the displaced transition molecule and the new active transition molecules. The small and large subunits can move one unit sideways relative to each other. Such movement is necessary following a change of direction of the computation. An incoming transition molecule **3** is approaching the active transition molecule **4** and the alphabet molecule to its right. The tape polymer can move left or right 1 unit, aligning the active transition molecule to the left or to the right side of the large tunnel. Such movement is necessary to accommodate consecutive transitions in the same direction. The hardware as well as the data tapes and the incoming transition molecule are shown.

Fig. 5G shows the mechanical implementation of the gating mechanisms, front (left) and back views. Five mechanisms in the small tunnel prevent erroneous transitions from occurring. All mechanisms are based on a spring-loaded bell crank/cam which is connected to a linkage which, in its free state, blocks passage of the approaching transition molecule. Each bell crank/cam checks for a certain condition and, if the condition is met, is rotated. The connected linkage then moves out of the way of the approaching transition molecule, essentially effecting a conformational change in the tunnel. Two mechanisms **1** and **2** detect that the (left or right) transition molecule is loaded with an alphabet molecule. Mechanism **3** detects that the recognition site of the incoming transition molecule matches the state side group of the active transition molecule and the alphabet symbol to its right. Additional two mechanisms check for the blank transition.

The computer operates in cycles (Fig. 6), processing one transition molecule per cycle. In each cycle, an incoming loaded transition molecule **2** that matches the current state and its adjacent alphabet monomer of the data polymer **1** becomes the new active transition molecule and its accompanying alphabet monomer is incorporated into the tape polymer via the intermediate **3**. This is achieved by displacing the currently active transition molecule and the matched alphabet monomer, effectively editing the tape polymer, and elongating the trace polymer by the displaced molecules to form the next configuration **4**. Specifically, when processing a left transition molecule, the computer moves left to accommodate the molecule, if necessary, and displaces the currently active transition molecule and the alphabet monomer to its left by the new molecule. The computer processes a right transition molecule similarly by moving right and displacing the alphabet monomer to the right of the active transition molecule.

The trace polymer created during the computation represents past state changes and head movements, as well as the symbols that were ''erased'' from the tape during each transition, and as such has several important advantages. First, the trace polymer renders the computer reversible. Because the trace polymer embodies a complete

record of the computation, a molecular implementation of the computer will be subject to the speed/energy consumption tradeoff of reversible devices. Second, computation traces, in general, and the trace polymer, in particular, enable many ''software'' program analysis and debugging tools,[24] which are critically needed for large-scale applications. Third, the trace polymer enables ''hardware'' error detection and correction. One expects that any biomolecular implementation of the computer may exhibit nonnegligible error rate. Such errors can be detected, and possibly also corrected, by cascading several computers along the same trace polymer, each detecting, and possibly also correcting, errors produced by its predecessor.

The most important property of the mechanical computer is that it is reactive:[25] it can have an ongoing, program-controlled, interaction with its environment. This capability is a result of the biologically inspired architecture of the computer rather than inherited from the theoretical Turing machine, which was conceived as a ''batch'' computing device that receives its input at the beginning of the computation and produces an output if and when the computation ends. The ribosome, for example, suspends the construction of a polypeptide chain when a required amino acid is unavailable. Similarly, this computer can be ''programmed'' to suspend until a specific molecule is available. The availability of such a control molecule can be tied to other relevant environmental conditions, thus triggering a computation only when these conditions prevail.

The Turing machine is a nondeterministic computing device in that it can make choices during a computation, and so is the mechanical computer. In a biomolecular implementation, this capability can be used to have the environment affect the course of a computation, based on the relative concentrations of molecules that enable one computational step compared with molecules enabling a different computational step. Using these two capabilities, the computer can be programmed so that both the timing and the course of a computation are affected and controlled by the biochemical environment.

The computer is endowed with an output device as follows. A simple extension to the Turing machine design is an instruction that erases the tape segment to the right of the read/write head. This instruction would mean in this context: ''cleave the tape polymer to the right of the active transition molecule and release this tape polymer segment to the environment.'' With this instruction, the computer can create and release any effectively computable polymer of alphabet monomers, in any number of copies, in the course of a computation. A cleaved tape polymer segment released by one computer can serve as the initial tape for the computation of another computer, or it can be ligated under certain conditions to the tape of another computer, thus enabling parallel processing, communication, and synchronization among multiply operating computers.

The computer design allows it to respond to the availability and to the relative concentrations of specific molecules in its environment and to construct program-defined polymers and release them into the environment. Hence if implemented using biomolecules, the computer can be part of biochemical pathways. In particular, given a biomolecular implementation of the computer that uses ribonucleic acids as alphabet monomers, one can envision how cleaved tape polymer segments can function as messenger-RNA, effecting program-directed synthesis of proteins in response to specific biochemical conditions within the cell. Such an implementation can provide a family of computing devices with broad biological and pharmaceutical applications.

Molecular Finite Automata

Two programmable, autonomous finite automata made of biomolecules were demonstrated by Benenson et al.[26,27] Both use a DNA molecule as input, DNA molecules as software, encoding the automaton transitions, and DNA-manipulating enzymes as hardware. The differences between the two are the source of energy for the computation and the reuse of software molecules. The first automaton relies on ATP as an energy source and consumes its software molecules during computation, while the second utilizes solely the energy stored in the chemical bonds of its DNA input molecule and its software molecules are reusable. While having similar logical structures, these versions differ in the implementation details. The design of the molecular finite automaton incorporates ideas from designs for molecular Turing machines.[15,16] The hardware of the first automaton consists of a mixture of the class IIS restriction nuclease *Fok*I, T4 DNA ligase, and ATP, while the second automaton utilizes only *Fok*I.

The structure of the latter automaton is shown in Fig. 7. Fig. 7A shows the encoding of *a*, *b*, and terminator (sense strands) and the <state, symbol> interpretation of exposed 4-nucleotide sticky ends, the leftmost representing the current symbol and the state S1, similarly the rightmost for S0. Fig. 7B shows the hardware: the *Fok*I restriction enzyme, which recognizes the sequence GGATG and cleaves 9 and 13 nucleotides apart on the $5' \rightarrow 3'$ and $3' \rightarrow 5'$ strands, respectively. The software comprises eight short double-stranded (ds) DNA molecules, the transition molecules encoding the eight possible transition rules (Fig. 7C). It consists of a <state, symbol> detector (light gray), a *Fok*I recognition site (dark gray), and a spacer (intermediate gray) of variable length that determines the

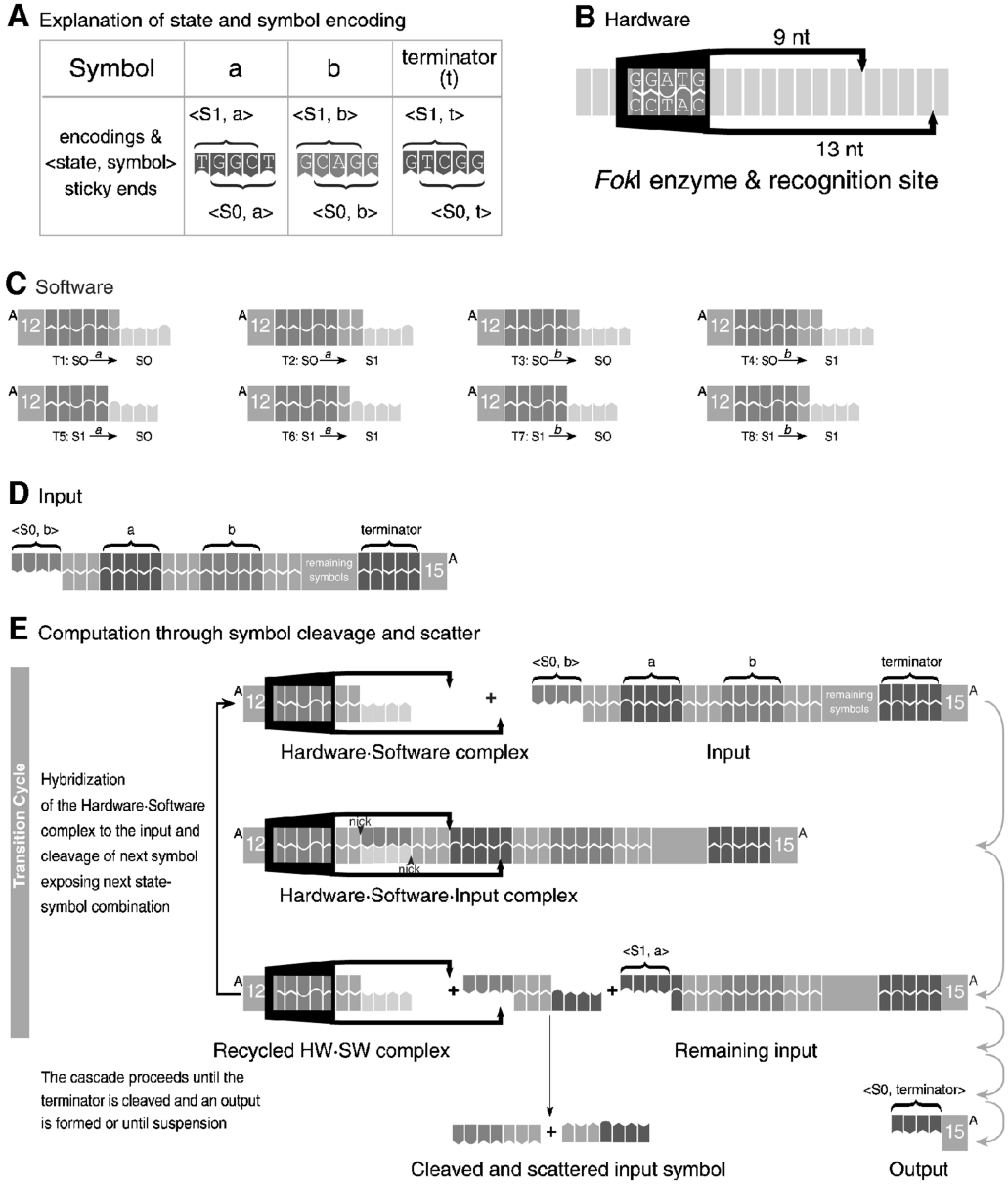

Fig. 7 Design details and mechanism of operation of the molecular automata of Benenson et al. (From Ref. [27] © PNAS.)

*Fok*I cleavage site inside the next symbol, which, in turn, defines the next state. Empty spacers effect S1 to S0 transition, single base-pair (bp) spacers maintain the current state, and 2-bp spacers transfer S0 to S1.

A dsDNA molecule encodes the initial state of the automaton and the input (Fig. 7D), with five to six base pairs (bp) coding for one input symbol (Fig. 7A), with the exposed sticky end at the 5′-terminus encoding the initial state and the first symbol. The ligase-based system may also contain ''peripherals,'' two output-detection molecules of different lengths, each of which can ligate selectively with a different output molecule to form an output-reporting molecule that indicates a final state and can be readily detected by gel electrophoresis. In the ATP-free system, the output is detected by examining the length of the remainder of a processed input molecule. The computation starts when the hardware, software, and input are all mixed together and runs autonomously, if possible till termination. The automaton processes the input as shown in Fig. 7E. The computation proceeds via a cascade of transition cycles. In each cycle, the sticky end of an applicable transition molecule may ligate to

the sticky end of the input molecule, detecting the current state and the current symbol. Alternatively, it may hybridize noncovalently. In both cases, the product is cleaved by *Fok*I inside the next symbol encoding, exposing a new four-nucleotide sticky end. The design of the transition molecules ensures that the encodings of the input symbols *a* and *b* are cleaved by *Fok*I at only two different ''frames'',[16] the leftmost frame encoding the state S1 and the rightmost frame encoding S0 (Fig. 7A). The exact next restriction site and thus the next internal state are determined by the current state and the size of the spacers (Fig. 7C, intermediate gray) in an applicable transition molecule. The computation proceeds until no transition molecule matches the exposed sticky end of the input or until the special terminator symbol is cleaved, forming an output molecule that has a sticky end encoding the final state. In a step extraneous to the computation and analogous to a ''print'' instruction of a conventional computer, this sticky end may ligate to one of two output detectors and the resultant output reporter may be identified by gel electrophoresis.

The ATP-free automaton has several advantages over the ligase-based version. First, the software molecule used in a transition is recycled because it undergoes no modification. Thus a finite number of software and hardware molecules may, in principle, process an input of any length. Second, much better performance characteristics may be achieved with the ATP-free automaton because the processing does not involve the usually slow ligation step. Using stoichiometric amounts of the software and the hardware molecules, it is possible to process a single symbol in a few seconds. On the other hand, the ligase-based automaton is less structurally restricted. It may utilize different SII-type restriction enzymes as a hardware, including those that require a covalently bonded substrate. More importantly, it was found[27] that the ability of *Fok*I to cleave DNA with its recognition and cleavage sites attached by sticky-end hybridization was limited to specific hybridization complexes. Long spacers and low GC content often resulted in cleaving only one of the input strands, producing a computationally illegal configuration. Correct performance was achieved with short spacers and high GC content of the sticky ends. The final design used the shortest possible spacers of 0, 1, and 2 bp (Fig. 7C), which dictated a particular symbol encoding (Fig. 7A) and the introduction of spacers between the symbols (Fig. 7D). Using ligase may relax some of these constraints.

These molecular automata may be viewed from two perspectives. On one hand, the computations were performed with bulk amounts of the input molecules. Each molecule was processed independently and in parallel, thus the inputs could potentially be distinct. The parallel character of the computation could be employed in a hypothetical process of screening of DNA libraries. Large libraries of molecules could be filtered through the same software, for a search of certain sequence feature. Traditional approach to the same problem would require (nonparallel) sequencing of the whole library and then running nonparallel computer algorithms on the sequences. To analyze parallel performance of our system, the cumulative number of unit operations in a unit time per unit volume was measured. This would represent an upper limit on the complexity of the libraries that could be analyzed. The parallel performance of the ligase-based version was in the order of 8.3×10^6 operations/sec/μL, while in the ATP-free case, the performance was improved almost 8000-fold and reached 6.6×10^{10} operations/sec/μL.

Another approach is to try and scale down the system to run it in a very small compartment such as living cell. Then the question is what are the minimal requirements from the operational system. It is not unfeasible that a mixture of a single input molecule, four software molecules, and one or two *Fok*I molecules could form a functioning computer while placed in a sufficiently small volume (to avoid dilution). While such possibility still requires experimental demonstration, it is possible to estimate its characteristics from the process performed in the bulk. Scaling down the concentrations, such a ''minimal computer'' would fit in a cube with a side length of 100 nm. The size of each component is in the range of several nanometers, with long inputs being tens of nanometers long. Such a computer would be a truly nanoscale computer. A computation on a single input molecule would proceed at a rate of 1000 sec per step in the ligase-based version and about 20 sec in the ATP-free version. While such rates seem slow compared with the electronic computers, they reflect the properties of biological systems. Once these computers would be able to operate in a cell, their performance would be competitive with respect to other cellular processes.

CONCLUSION

The notion of a biomolecular computing machine has evolved gradually over the past decades. Theoretical designs proposed for such machines eventually led to simple molecular computing machines functioning in the test tube. The field may develop in several directions. First, more complex computing machines could be designed and built. This includes general finite automata, string transducers, stack automata, and, ultimately, the Turing machine. Currently, progress in this direction

seems to be hampered by the lack of DNA- and RNA-manipulating enzymes; we hope that an eventual progress in enzyme engineering may supply the tools required to develop more complex machines. Another important issue relevant to many machine designs is symbol encoding. Current experimental realizations utilize artificial alphabet of predesigned DNA sequences. However, the computer should ''understand'' natural alphabets of either single nucleotides or amino acid codons to be biologically relevant. Designing even the simplest finite automaton that would operate on an arbitrary DNA sequence remains a major challenge. A third future direction is a search for application that would clearly demonstrate qualitative edge of a molecular computer over competing technologies.

We believe that the application potential of autonomous biomolecular computers is not to surpass electronic computers with performance; in fact, it is hard to believe they ever will. The advantage is that biomolecular computers process information encoded in molecules rather than in electric signals. Any *direct* computing over biomolecular inputs could only be performed by the computers of the same format, i.e., made of biomolecules. As we already mentioned, running sequence analysis algorithms on DNA libraries without actually sequencing all library members could be conveniently performed by a molecular state machine whose alphabet is composed of single nucleotides or codons. Another broad range of application may emerge once the molecular computer is successfully ''plugged into'' cellular molecular environment. By ''plugging into'' we mean that some of the computer components would be able to respond to certain changes in the environment, affecting the result of the computation. While the most obvious component to communicate with the environment seems to be the software, both the hardware and input could be affected as well. Once the communication between the intracellular compounds and the computer is established, the computer may, in principle, perform complex analysis of the environmental conditions. The complexity of such analysis would increase with the complexity of the computing machine and the sensitivity of the communication channels. However, even the simplest finite automata seem to provide enough computational power to make rather complex diagnostics.

REFERENCES

1. Adleman, L.M. Molecular computation of solutions to combinatorial problems. Science **1994**, *266* (5187), 1021–1024.
2. Kari, L. DNA computing: Arrival of biological mathematics. Math. Intell. **1997**, *19* (2), 9–22.
3. Lipton, R.J. DNA solution of hard computational problem. Science **1995**, *268* (5210), 542–545.
4. Ouyang, Q.; Kaplan, P.D.; Liu, S.; Libchaber, A. DNA solution of the maximal clique problem. Science **1997**, *278* (5337), 446–449.
5. Faulhammer, D.; Cukras, A.R.; Lipton, R.J.; Landweber, L.F. Molecular computation: RNA solutions to chess problems. Proc. Natl. Acad. Sci. U. S. A. **2000**, *97* (4), 1385–1389.
6. Winfree, E. On the Computational Power of DNA Annealing and Ligation. In *DNA Based Computers: Proceedings of the DIMACS Workshop, April 4, 1995, Princeton University*; Lipton, R.J., Baum, E.B., Eds.; American Mathematical Society: Providence, RI, 1996; 199–221.
7. Winfree, E. Algorithmic Self Assembly of DNA. Ph.D Thesis; Caltech, 1998.
8. Winfree, E.; Liu, F.; Wenzler, L.; Seeman, N.C. Design and self-assembly of two-dimensional DNA crystals. Nature **1998**, *394* (6693), 539–544.
9. Mao, C.; LaBean, T.H.; Reif, J.H.; Seeman, N.C. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature **2000**, *407* (6803), 493–496.
10. Wang, H. Proving theorems by pattern recognition. Part II'. Bell Syst. Tech. J. **1961**, *40*, 1–41.
11. Feng, L.; Park, S.; Reif, J.; Yan, H. A two-state DNA lattice switched by DNA nanoactuator. Angew. Chem. **2003**, *115* (36), 4478–4482.
12. Yan, H.; Park, S.; Finkelstein, G.; Reif, J.; LaBean, T. DNA-templated self-assembly of protein arrays and highly conductive nanowires. Science **2003**, *301* (5641), 1882–1884.
13. Bennett, C.H. The thermodynamics of computation—A review. Int. J. Theor. Phys. **1982**, *21* (12), 905–940.
14. Shapiro, E. A Mechanical Turing Machine: Blueprint for a Biomolecular Computer. In *Proc. 5th Int. Meeting on DNA Based Computers*; Winfree, E., Giffrod, D., Eds.; DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society: Providence, RI, 1995; 229–230.
15. Shapiro, E.; Karunaratne, K.S.G. Method and System of Computing Similar to a Turing Machine. US Patent 6,266,569, 2001.
16. Rothemund, P.W.K. A DNA and Restriction Enzyme Implementation of Turing Machine. In *DNA Based Computers: Proceedings of the DIMACS Workshop, April 4, 1995, Princeton University*; Lipton, R.J., Baum, E.B., Eds.; American Mathematical Society: Providence, RI, 1996; 75–119.
17. Smith, W.D. DNA Computers in vivo and in vitro. In *DNA Based Computers: Proceedings of the*

DIMACS Workshop, April 4, 1995, Princeton University; Lipton, R.J., Baum, E.B., Eds.; American Mathematical Society: Providence, RI, 1996; 121–185.

18. Garzon, M.; Gao, Y.; Rose, J.A.; Murphy, R.C.; Deaton, R.; Franceschetti, D.R.; Stevens, S.E. In vitro Implementation of Finite-State Machines. In *Automata Implementation: Lecture Notes in Computer Science 1436*; Wood, D., Yu, S., Eds.; Springer: Berlin, 1998; 56–74.
19. Sakamoto, K.; Kiga, D.; Komiya, K.; Gouzu, H.; Yokoyama, S.; Ikeda, S.; Sugiyama, H.; Hagiya, M. State transitions by molecules. Biosystems **1999**, *52* (1–3), 81–91.
20. Sakamoto, K.; Gouzu, H.; Komiya, K.; Kiga, D.; Yokoyama, S.; Yokomori, T.; Hagiya, M. Molecular computation by DNA hairpin formation. Science **2000**, *288* (2469), 1223–1226.
21. Turing, A.M. On computable numbers, with an application to the Entcheidung problem. Proc. Lond. Math. Soc., II Ser. **1936**, *42*, 230–265.
22. Hopcroft, J.E.; Motwani, R.; Ullmann, J.D. *Introduction to Automata Theory, Languages, and Computation,* 2nd Ed.; Addison Wesley: Boston, 2000.
23. Alberts, B.; Johnson, A.; Lewis, J.; Raff, M.; Roberts, K.; Walter, P. *Molecular Biology of the Cell,* 4th Ed.; Garland: New York, 2002.
24. Shapiro, E. *Algorithmic Program Debugging*; MIT Press: Cambridge, MA, 1982.
25. Harel, D.; Pnueli, A. On the Development of Reactive System. In *Logics and Models of Concurrent Systems*; Apt, K.R., Ed.; Springer-Verlag: New York, 1985; 477–498.
26. Benenson, Y.; Paz-Elizur, T.; Adar, R.; Keinan, E.; Livneh, Z.; Shapiro, E. Programmable and autonomous computing machine made of biomolecules. Nature **2001**, *414* (6862), 430–434.
27. Benenson, Y.; Adar, R.; Paz-Elizur, T.; Livneh, Z.; Shapiro, E. DNA molecule provides a computing machine with both data and fuel. Proc Natl. Acad. Sci. U. S. A. **2003**, *100* (5), 2191–2196.

Marcel Dekker, Inc.
270 Madison Avenue, New York, New York 10016