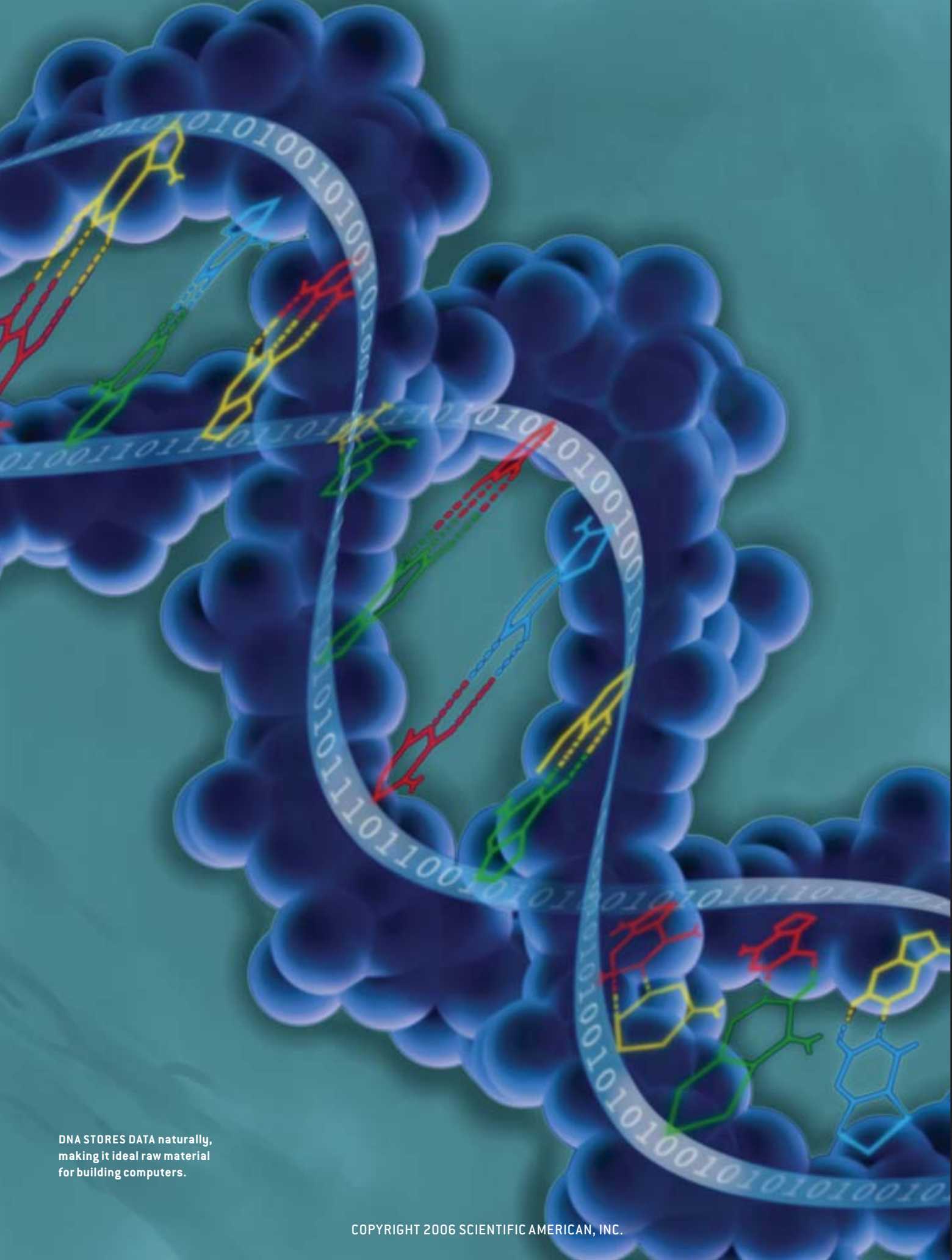


DNA STORES DATA naturally, making it ideal raw material for building computers.

COPYRIGHT 2006 SCIENTIFIC AMERICAN, INC.

BRINGING DNA COMPUTERS TO LIFE

Tapping the computing power of biological molecules gives rise to tiny machines that can speak directly to living cells

By Ehud Shapiro and Yaakov Benenson

When British mathematician Alan Turing conceived the notion of a universal programmable computing machine, the word "computer" typically referred not to an object but to a human being. It was 1936, and people with the job of computer, in modern terms, crunched numbers. Turing's design for a machine that could do such work instead—one capable of computing any computable problem—set the stage for theoretical study of computation and remains a foundation for all of computer science. But he never specified what materials should be used to build it.

Turing's purely conceptual machine had no electrical wires, transistors or logic gates. Indeed, he continued to imagine it as a person, one with an infinitely long piece of paper, a pencil and a simple instruction book. His tireless computer would read a symbol, change the symbol, then move on to the next symbol, according to its programmed rules, and would keep doing so until no further rules applied. Thus, the electronic computing machines made of metal and vacuum tubes that emerged in the 1940s and later evolved silicon parts may be the only "species" of nonhuman computer most people have ever encountered, but theirs is not the only possible form a computer can take.

Living organisms, for instance, also carry out complex physical processes under the direction of digital information. Biochemical reactions and ultimately an entire organism's operation are ruled by instructions stored in its genome, encoded in sequences of nucleic acids. When the workings of biomolecular machines inside cells that process DNA and RNA are compared to Turing's machine, striking similarities emerge: both systems process information stored in a string of symbols taken from a fixed alphabet, and both operate by moving step by step along those strings, modifying or adding symbols according to a given set of rules.

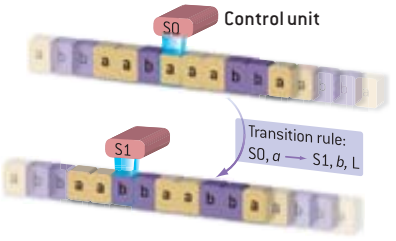
These parallels have inspired the idea that biological molecules could one day become the raw material of a new computer species. Such biological computers would not necessarily offer greater power or performance in traditional computing tasks. The speed of natural molecular machines such as the ribosome is only hundreds of operations a second, compared

MATT COLLINS

COPYRIGHT 2006 SCIENTIFIC AMERICAN, INC.

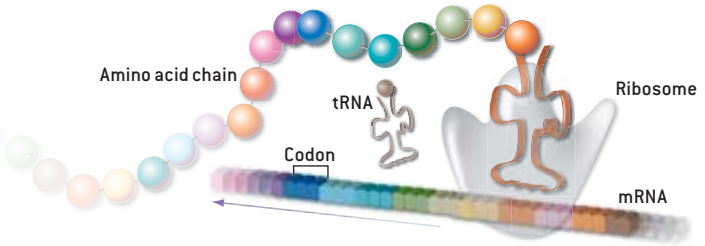
COMPUTING MACHINES: CONCEPTUAL AND NATURAL

Mathematician Alan Turing envisioned the properties of a mechanical computer in 1936, long before molecule-scale machines within cells could be seen and studied. As the workings of nature's tiny automata were later revealed, striking similarities to Turing's concept emerged: both systems store information in strings of symbols, both process these strings in stepwise fashion, and both modify or add symbols according to fixed rules.



TURING MACHINE

This hypothetical device operates on an information-encoding tape bearing symbols such as "a" and "b." A control unit with read/write ability processes the tape, one symbol position at a time, according to instructions provided by transition rules, which note the control unit's own internal state. Thus, the transition rule in this example dictates that if the control unit's state is 0 (S0), and the symbol read is *a*, then the unit should change its state to 1 (S1), change the symbol to *b* and move one position to the left (L).



BIOLOGICAL MACHINE

An organelle found in cells, the ribosome reads information encoded in gene transcripts known as messenger RNAs (mRNAs) and translates it into amino acid sequences to form proteins. The symbolic alphabet of mRNA is made up of nucleotide trios called codons, each of which corresponds to a specific amino acid. As the ribosome processes the mRNA strand, one codon at a time, helper molecules called transfer RNAs (tRNAs) deliver the correct amino acid. The tRNA confirms the codon match, then releases the amino acid to join the growing chain.

with billions of gate-switching operations a second in some electronic devices. But the molecules do have a unique ability: they speak the language of living cells.

The promise of computers made from biological molecules lies in their potential to operate within a biochemical environment, even within a living organism, and to interact with that environment through inputs and outputs in the form of other biological molecules. A biomolecular computer might act as an autonomous "doctor" within a cell, for example. It could sense signals from the environment indicating disease, process them using its preprogrammed medical knowledge, and output a signal or a therapeutic drug.

Over the past seven years we have been working toward realizing this vision. We have already succeeded in creating a biological automaton made of DNA and proteins able to diagnose in a test tube the molecular symptoms of certain cancers and "treat" the disease by releasing a therapeutic molecule. This proof of concept was exciting, both because it has potential future medical applications and because it is not at all what we originally set out to build.

Overview/*Living Computers*

- Natural molecular machines process information in a manner similar to the Turing machine, an early conceptual computer.
- A Turing-like automaton built from DNA and enzymes can perform computations, receive input from other biological molecules and output a tangible result, such as a signal or a therapeutic drug.
- This working computer made from the molecules of life demonstrates the viability of its species and may prove a valuable medical tool.

Models to Molecules

ONE OF US (Shapiro) began this research with the realization that the basic operations of certain biomolecular machines within living cells—recognition of molecular building blocks, cleavage and ligation of biopolymer molecules, and movement along a polymer—could all be used, in principle, to construct a universal computer based on Turing's conceptual machine. In essence, the computational operations of such a Turing machine would translate into biomolecular terms as one "recognition," two "cleavages," two "ligations," and a move to the left or right.

Charles Bennett of IBM had already made similar observations and proposed a hypothetical molecular Turing machine in 1982. Interested in the physics of energy consumption, he speculated that molecules might one day become the basis of more energy-efficient computing devices [see "The Fundamental Physical Limits of Computation," by Charles H. Bennett and Rolf Landauer; SCIENTIFIC AMERICAN, July 1985].

The first real-world demonstration of molecules' computational power came in 1994, when Leonard M. Adleman of the University of Southern California used DNA to solve a problem that is always cumbersome for traditional computer algorithms. Known as the Hamiltonian path or the traveling salesman problem, its goal is to find the shortest path among cities connected by airline routes that passes through every city exactly once. By creating DNA molecules to symbolically represent the cities and flights and then combining trillions of these in a test tube, he took advantage of the molecules' pairing affinities to achieve an answer within a few minutes [see

LUCY READING-IKKANDA

COPYRIGHT 2006 SCIENTIFIC AMERICAN, INC.

"Computing with DNA," by Leonard M. Adleman; Scientific American, August 1998]. Unfortunately, it took him considerably longer to manually fish the molecules representing the correct solution out of the mixture using the laboratory tools available to him at the time. Adleman looked forward to new technologies that would enable the creation of a more practical molecular computer.

"In the future, research in molecular biology may provide improved techniques for manipulating macromolecules," Adleman wrote in a seminal 1994 scientific paper describing the DNA experiment. "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 which act upon it."

Devising a concrete logical design for just such a device, one that could function as the fundamental "operational specification" for a broad class of future molecular computing machines, became Shapiro's goal. By 1999 he had a mechanical model of the design made from plastic parts. We then joined forces to translate that model into real molecules.

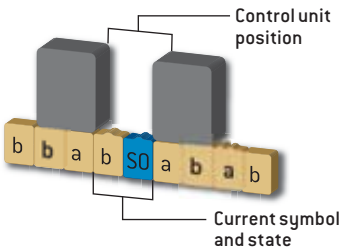
Rather than attacking the ultimate challenge of building a full-fledged molecular Turing machine head-on, however, we agreed to first attempt a very simplified Turing-like machine known as a finite automaton. Its sole job would be to determine whether a string of symbols or letters from a two-letter alphabet, such as "a" and "b," contained an even number of *b*'s. This task can be achieved by a finite automaton with just two states and a "program" consisting of four statements called transition rules. One of us (Benenson) had the idea to use a double-stranded DNA molecule to represent the input string, four more short double-stranded DNA molecules to represent the automaton's transition rules, or "software," and two natural DNA-manipulating enzymes, *Fok*I and ligase, as "hardware."

The main logical problem we had to solve in its design was how to represent the changing intermediate states of the computation, which consist of the current internal state of the automaton and a pointer to the symbol in the input string being

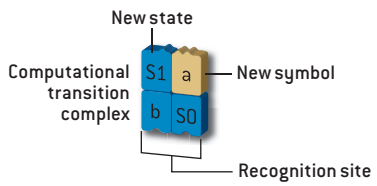
MOLECULAR TURING MACHINE MODEL

A Turing machine made of biomolecules would employ their natural ability to recognize symbols and to join molecular subunits together or cleave their bonds. A plastic model built by one of the authors (*right*) serves as a blueprint for such a system. Yellow "molecule" blocks carry the symbols. Blue software molecules indicate a machine state and define transition rules. Protrusions on the blocks physically differentiate them.

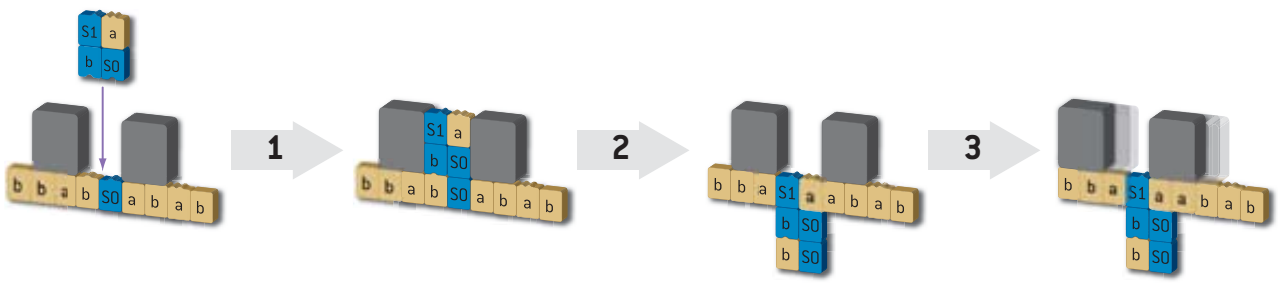
HOW IT WORKS



The machine operates on a string of symbol molecules. In its control unit position at the center, both a symbol and the machine's current state are defined.



One "computational transition" is represented by a molecule complex containing a new state and symbol for the machine and a recognition site to detect the current state and symbol. The example shown represents a transition rule: "If current state is S0 and current symbol is *b*, change state to S1 and symbol to *a*, then move one step to the left."



A free-floating computational transition complex slides into the machine's control unit (*1*). The molecule complex binds to and then displaces the current symbol and state (*2*). The control unit can move one position to the left to accommodate another transition complex (*3*). The process repeats indefinitely with new states and symbols as long as transition rules apply.

LUCY READING-IKKANDA (*illustrations*); WEIZMANN INSTITUTE OF SCIENCE (*photograph*)

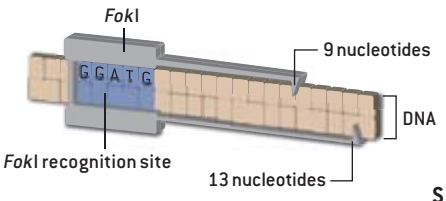
COPYRIGHT 2006 SCIENTIFIC AMERICAN, INC.

BUILDING A MOLECULAR AUTOMATON

Because living organisms process information, their materials and mechanisms lend themselves readily to computing. The DNA molecule exists to store data, written in an alphabet of nucleotides. Cellular machinery reads and modifies that information using enzymes and other molecules. Central to this operating system are chemical affinities among molecules allowing them to recognize and bind with one another. Making molecules into a Turing-like device, therefore, means translating his concepts into their language.

A simple conceptual computer called a finite automaton can move in only one direction and can read a series of symbols, changing its internal state according to transition rules. A two-state automaton could thus answer a yes-no question by alternating between states designated 1 and 0. Its state at the end of the calculation represents the result.

Raw materials for a molecular automaton include DNA strands in assorted configurations to serve as both input and software and the DNA-cleaving enzyme *Fok*I as hardware. Nucleotides, whose names are abbreviated A, C, G and T, here encode both symbols and the machine's internal state.



HARDWARE

The *Fok*I enzyme (*gray*) always recognizes the nucleotide sequence GGATG (*blue*) and snips a double DNA strand at positions 9 and 13 nucleotides downstream of that recognition site.

SOFTWARE

Transition rules are encoded in eight short double-stranded DNA molecules containing the *Fok*I recognition site (*blue*), followed by spacer nucleotides (*green*) and a single-stranded sticky end (*yellow*) that will join to its complementary sequence on an input molecule.

SYMBOL AND STATE

Combinations of symbols *a*, *b* or *terminator* (*t*) and machine states 1 or 0 are represented by four-nucleotide sequences. Depending on how the five-nucleotide sequence TGGCT is cleaved into four nucleotides, for example, it will denote symbol *a* and a state of either 1 or 0.

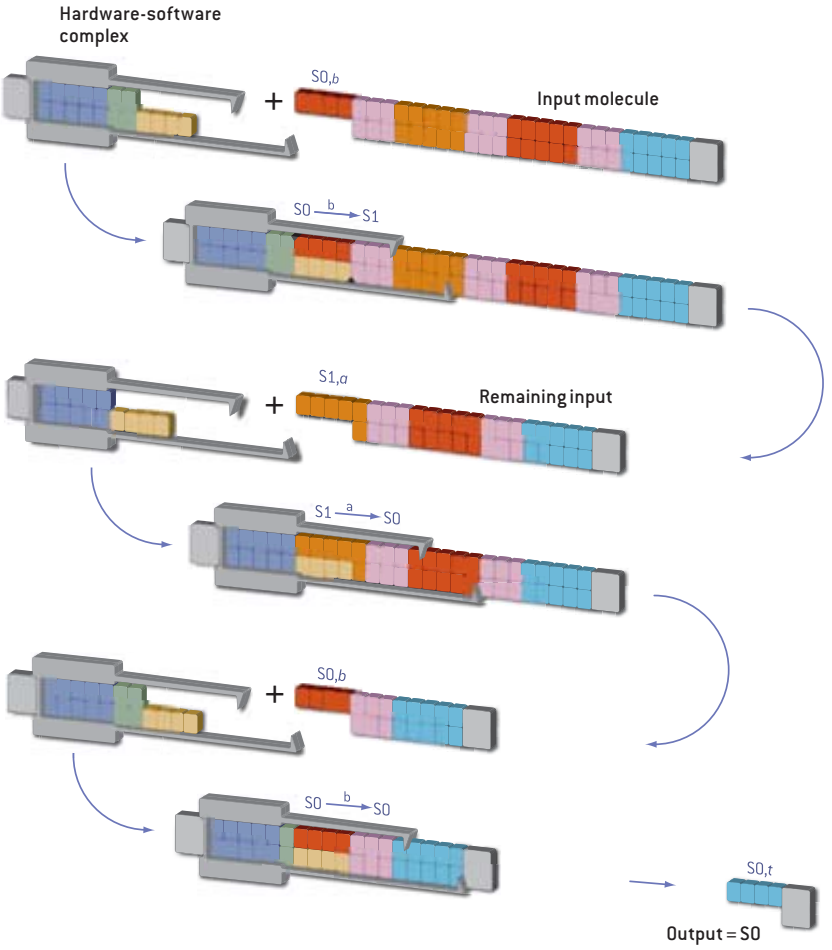
AUTONOMOUS COMPUTATION

A hardware-software complex recognizes its complementary state/symbol combination on the input molecule. The molecules join to form a hardware-software-input complex, then *Fok*I cleaves the input molecule to expose the next symbol.

A new hardware-software complex recognizes the next state and symbol on what remains of the input molecule.

Reactions continue until no rule applies or the terminator symbol is revealed.

In this example, computational cleavages leading to the final output (*far right*) have produced a four-nucleotide terminator symbol indicating a machine state of 0, the calculation's result.



LUCY READING-IKKANDA

COPYRIGHT 2006 SCIENTIFIC AMERICAN, INC.

processed. We accomplished this with a neat trick: in each step of the computation the enzymatic hardware was actually "digesting" the input molecule, cleaving the current symbol being processed and exposing the next one. Because the symbol could be cleaved in two different locations, each resulting version of it could represent, in addition to the symbol itself, one of two possible states of the computation. Interestingly, we discovered later that this last element was similar to a design that Paul Rothemund, a former student of Adleman, once proposed for a molecular Turing machine.

Remarkably, the resulting computer that our team announced in 2001 was autonomous: once the input, software and hardware molecules were placed in an appropriate buffer solution in a test tube, computation commenced and proceeded iteratively to completion without any human intervention.

As we tested this system, we also realized that it not only solved the original problem for which we had intended it—determining whether a symbol occurs an even number of times in a string—it could do more. A two-state, two-symbol finite automaton has eight possible symbol-state-rule combinations (2^3), and because our design was modular, all eight possible transition rules could be readily implemented using eight different transition molecules. The automaton could therefore be made to perform different tasks by choosing a different "program"—that is, a different mix of transition molecules.

In trying a variety of programs with our simple molecular automaton, we also found a way of further improving its performance. Among our tests was an omission experiment, in which the automaton's operation was evaluated with one molecular component removed at a time. When we took away ligase, which seals the software molecule to the input molecule to enable its recognition and cleavage by the other enzyme, *Fok*I, the computation seemed to make some progress nonetheless. We had discovered a previously unknown ability of *Fok*I to recognize and cleave certain DNA sequences, whether or not the molecule's two strands were sealed together.

The prospect of removing ligase from our molecular computer design made us quite happy because it would immediately reduce the required enzymatic hardware by 50 percent. More important, ligation was the only energy-consuming operation in the computation, so sidestepping it would allow the computer to operate without an external source of fuel. Finally, eliminating the ligation step would mean that software molecules were no longer being consumed during the computation and could instead be recycled.

The ligase-free system took our group months of painstaking effort and data analysis to perfect. It was extremely inefficient at first, stalling out after only one or two computational steps. But we were driven by both the computational and biochemical challenges, and with help and advice from Rivka Adar and other colleagues, Benenson finally found a solution. By making small but crucial alterations to the DNA sequences used in our automaton, we were able to take advantage of *Fok*I's hitherto unknown capability and achieve a quantum leap in the computer's performance. By 2003 we had an autonomous, programmable computer that could use its input molecule as its sole source of fuel [*see box on opposite page*]. In principle, it could therefore process any input molecule, of any length, using a fixed number of hardware and software molecules without ever running out of energy.

And yet from a computational standpoint, our automaton still seemed like a self-propelled scooter compared with the Rolls-Royce of computers on which we had set our sights: the biomolecular Turing machine.

DNA Doctor

BECAUSE THE TWO-STATE finite automaton was too simple to be of any use in solving complex computational problems, we considered it nothing more than an interesting demonstration of the concept of programmable, autonomous biomolecular computers, and we decided to move on. Focusing our efforts for a while on trying to build more complicated automata, however, we soon ran up against the problem recognized by Adleman: the "designer enzymes" he had yearned for a decade earlier still did not exist.

No known naturally occurring enzyme or enzyme complex can perform the specific recognitions, cleavages and ligations, in sequence and in tandem, with the flexibility needed to realize the Turing machine design. Natural enzymes would have to be customized or entirely new synthetic enzymes engineered. Because science does not yet have this ability, we found ourselves with a logical design specification for a biomolecular Turing machine but forced to wait until the parts needed to build it are invented.

That is why we returned to our two-state automaton to see if we could at least find something useful for it to accomplish. With medical applications already in mind, we wondered if the device might be able to perform some kind of simple diagnosis, such as determining whether a set of conditions representing a particular disease is present.

For this task, just two states suffice: we called one state "yes" and the other "no." The automaton would begin the computation in the *yes* state and check one condition at a time. If a condition on its checklist were present, the *yes* state would hold, but if any condition were not present, the automaton would change to the *no* state and remain that way for the rest of the computational process. Thus, the computation would

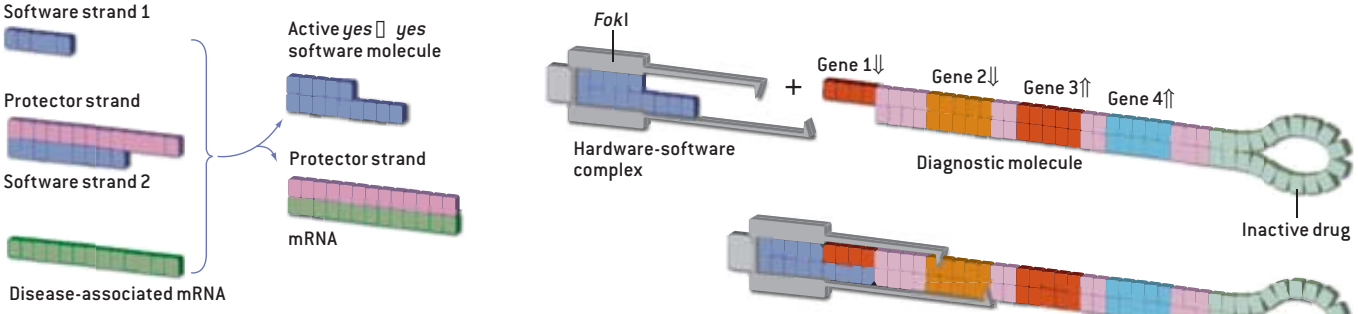
THE AUTHORS

EHUD SHAPIRO and *YAAKOV BENENSON* began collaborating to build molecular automata in 1999. Shapiro is a professor in the departments of computer science and biological chemistry at the Weizmann Institute of Science in Rehovot, Israel, where he holds the Harry Weinrebe Professorial Chair. He was already an accomplished computer scientist and software pioneer with a growing interest in biology in 1998 when he first designed a model for a molecular Turing machine. Benenson, just completing a master's degree in biochemistry at the Technion in Haifa, became Shapiro's Ph.D. student the following year. Now a fellow at Harvard University's Bauer Center for Genomics Research, Benenson is working on new molecular tools to probe and affect live cells.

COPYRIGHT 2006 SCIENTIFIC AMERICAN, INC.

DNA DOCTOR

Having shown that an automaton made from DNA and enzymes can perform abstract yes-or-no computations, the authors sought to give the device a practical query that it might face inside a living cell: Are indicators of a disease present? If the answer is yes, the automaton can output an active drug treatment. The basic computational concept is unchanged from the earlier design: complexes of "software" transition molecules and enzymatic "hardware" process symbols within a diagnostic molecule, cleaving it repeatedly to expose subsequent symbols. In addition, the new task requires a means for disease indicators to create input for the computation and mechanisms for confirming the diagnosis and delivering treatment.



INPUT

Gene transcripts called messenger RNAs (mRNAs) serve as disease indicators. By interacting with software molecules, mRNAs influence which of them is ultimately used in the computation. In this example, the two strands of a *yes* → *yes* transition molecule start out separated, with one bound to a single protector strand. The protector has a strong affinity for the disease-associated mRNA, however. If that mRNA is present, the protector will abandon its software strand to bind to the mRNA. The single software strands will then bind to one another, forming an active *yes* → *yes* transition molecule.

COMPUTATION

Complexes of transition-molecule software and *Fok*I enzymatic hardware process a series of symbols within the diagnostic molecule that represent underactivity (⇓) or overactivity (⇑) by specific genes. The automaton starts the computation in a *yes* state, and if all disease indicators are present, it produces a positive diagnosis. If any symptom is missing, the automaton transitions to *no* and remains in that state.

end in *yes* only if all the disease conditions held, but if one condition were not met the "diagnosis" would be negative.

To make this logical scheme work, we had to find a way to connect the molecular automaton to its biochemical environment so that it could sense whether specific disease conditions were present. The general idea that the environment could affect the relative concentrations of competing transition molecules—and thus affect the computation—had already been suggested in the blueprint for the molecular Turing machine. To apply this principle to sense disease symptoms, we had to make the presence or absence of a disease indicator a determinant of the concentration of software molecules that testify for the symptom.

Many cancers, for example, are characterized by abnormal levels of certain proteins in the cell as a result of specific genes either overexpressing or underexpressing their encoded protein. When a gene is expressed, enzymes in the cell's nucleus copy its sequence into an RNA version. This molecular transcript of the gene, known as messenger RNA (mRNA), is then read by a structure called the ribosome that translates the RNA sequence into a string of amino acids that will form the protein. Thus, higher- or lower-than-normal levels of specific mRNA transcripts can reflect gene activity.

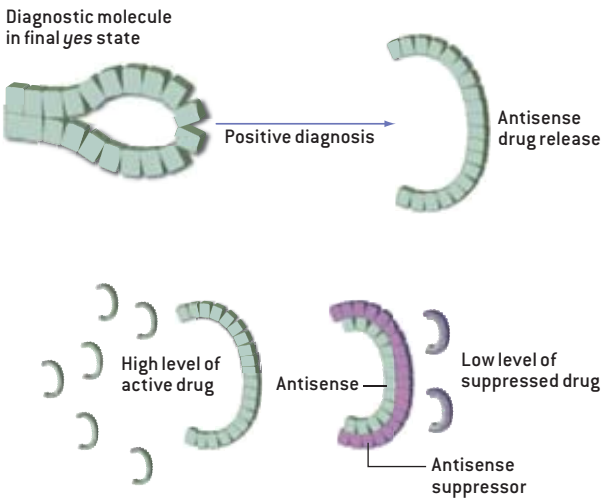
Benenson devised a system wherein some transition molecules would preferentially interact with these mRNA sequences. The interaction, in turn, would affect the transition molecules' ability to participate in the computation. A high level of mRNA representing a disease condition would cause the *yes* → *yes* transition molecules to predominate, increasing the probability that the computer would find the symptom to be present [*see box above*]. In practice, this system could be applied to any disease associated with abnormal levels of proteins resulting from gene activity, and it could also be adapted to detect harmful mutations in mRNA sequences.

Once we had both an input mechanism that could sense disease symptoms and the logical apparatus to perform the diagnosis, the next question became, What should the computer do when a disease is diagnosed? At first, we considered having it produce a visible diagnostic signal. In the molecular world, however, producing a signal and actually taking the next logical step of administering a drug are not that far apart. Binyamin Gil, a graduate student on our team, proposed and implemented a mechanism that enables the computer to release a drug molecule on positive diagnosis.

Still, our plan was not complete. Perhaps the central question in computer hardware design is how to build a reliable system from unreliable components. This problem is not unique to biological computers—it is an inherent property of complex systems; even mechanical devices become more unreliable as scale diminishes and the number of components increases. In our case, given the overall probabilistic nature of the computation and the imprecise behavior of biomolecular elements, some computations would inevitably end with a positive diagnosis even if several or all of the disease symp-

LUCY READING-IKKANDA

COPYRIGHT 2006 SCIENTIFIC AMERICAN, INC.



OUTPUT

After a positive diagnosis, final cleavage of the diagnostic molecule releases the treatment, a single-stranded so-called antisense DNA molecule (*top*). To compensate for diagnostic errors, the authors also created negative versions of the diagnostic molecules to perform parallel computations. When disease indicators are absent, these automata release a drug suppressor. With thousands of both types of diagnostic molecules computing simultaneously, the majority will make the correct diagnosis, and either the antisense molecule will outnumber its suppressors (*bottom*), or vice versa.

toms were absent, and vice versa. Fortunately, this probabilistic behavior is measurable and repeatable, so we could compensate for it with a system of checks and balances.

We created two types of computation molecules: one designed to release a drug when the computation terminates in the *yes* state, the other to release a suppressor of that same drug when the computation terminates in *no*. By changing the relative concentrations of the two types of molecules, we could have fine control over the threshold of diagnostic certainty that would trigger administration of an active drug.

Human physicians make this kind of decision whenever they weigh the risk to a patient of a possible disease against the toxicity of the treatment and the certainty of the diagnosis. In the future, if our molecular automaton is sent on a medical mission, it can be programmed to exercise similar judgment.

Dawn of a New Species

AS IT TURNED OUT, our simple scooter carried us much further than we had believed it could and in a somewhat different direction than we had first imagined. So far our biomolecular computer has been demonstrated only in a test tube. Its biological environment was simulated by adding different concentrations of RNA and DNA molecules and then placing all the automaton components in the same tube. Now our goals are to make it work inside a living cell, to see it compute inside the cell and to make it communicate with its environment.

Just delivering the automaton into the cell is challenging because most molecular delivery systems are tailored for either DNA or protein. Our computer contains both, so we are trying to find ways to administer these molecules in tandem. Another hurdle is finding a means of watching all aspects of the computation as they occur within a cell, to confirm that the automaton can work without the cell's activities disrupting computational steps or the computer's components affecting cellular behavior in unintended ways. And finally, we are exploring alternative means of linking the automaton to its environment. Very recent cancer research suggests that microRNAs, small molecules with important regulatory functions inside cells, are better indicators of the disease, so we are redesigning our computer to "talk" to microRNA instead of mRNA.

Although we are still far from applying our device inside living cells, let alone in living organisms, we already have the important proof of concept. By linking biochemical disease symptoms directly with the basic computational steps of a molecular computer, our test-tube demonstration affirmed that an autonomous molecular computer can communicate with biological systems and perform biologically meaningful assessments. Its input mechanism can sense the environment in which it operates; its computation mechanism can analyze that environment; and its output mechanism can affect the environment in an intelligent way based on the result of its analysis.

Thus, our automaton has delivered on the promise of biomolecular computers to enable direct interaction with the biochemical world. It also brings computational science full circle back to Turing's original vision. The first computing machines had to deviate from his concept to accommodate the properties of electronic parts. Only decades later, when molecular biologists began revealing the operations of tiny machines inside living cells did computer scientists recognize a working system similar to Turing's abstract idea of computation.

This is not to suggest that molecules are likely to replace electronic machines for all computational tasks. The two computer species have different strengths and can easily coexist. Because biomolecules can directly access data encoded in other biomolecules, however, they are intrinsically compatible with living systems in a way that electronic computers will never be. And so we believe our experiments suggest that this new computer species is of fundamental importance and will prove itself valuable for a wide range of applications. The biomolecular computer has come to life. SA

MORE TO EXPLORE

A Mechanical Turing Machine: Blueprint for a Biomolecular Computer. Presented by Ehud Shapiro at the 5th International Meeting on DNA Based Computers, Massachusetts Institute of Technology, June 14–15, 1999. **www.weizmann.ac.il/udi/press**

Programmable and Autonomous Computing Machine Made of Biomolecules. Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh and E. Shapiro in *Nature,* Vol. 414, pages 430–434; November 22, 2001.

An Autonomous Molecular Computer for Logical Control of Gene Expression. Y. Benenson, B. Gil, U. Ben-Dor, R. Adar and E. Shapiro in *Nature,* Vol. 429, pages 423–429; May 27, 2004.

COPYRIGHT 2006 SCIENTIFIC AMERICAN, INC.