COMPUTING COMES TO LIFE

Brian Hayes

A reprint from

American Scientist

the magazine of Sigma Xi, the Scientific Research Society

Volume 89, Number 3
May–June, 2001
pages 204–208

COMPUTING COMES TO LIFE

Brian Hayes

People have been recruiting other species to serve human needs for at least 10,000 years. We have turned plants into crops and animals into beasts of burden; even microorganisms have been pressed into service as fermenters. Yet until now no nonhuman species has ever been harnessed to do intellectual work on our behalf. That could change. Biologists and computer scientists have designed digital logic gates based on the metabolism of living cells, with the aim of eventually building a computer out of colonies of *Escherichia coli* or some other single-celled organism. But perhaps *build* is the wrong verb here; the plan is to *grow* or *breed* or *culture* a computer.

The idea of a bacterial computer is not in itself quite so outlandish as it may seem on first acquaintance. In principle, computing machines can be made out of almost anything, from billiard balls to Tinker Toys, and there is no reason that lipid sacs of proteins and nucleic acids should not also qualify as computer building blocks. From the lofty and austere perspective of computer science, an agar plate coated with microscopic bacteria is not much different from a silicon wafer etched with microscopic transistors. If the components can store and manipulate information in a few basic ways, they can compute.

So much for the lofty and austere view of computer science; but there is also computer engineering to be considered, and the questions asked in that discipline are more down to earth. Can living logic gates be strung together in networks large enough to perform an *interesting* computation? Can they run fast enough to complete a task within a human lifetime? Can they be made reliable enough to produce consistent and correct answers? Can biocomputer engineers cope with all the distinctive failure modes of living organisms—disease, predation, parasitism, senescence, death? (In this context the threat of a computer virus is more than a metaphor!) It's fair to say that practical applications of biological computers are a long way off. And yet skeptics might keep in mind that the historical record of domestications is a vast catalogue of unlikely-seeming successes.

Brian Hayes is Senior Writer for American Scientist. *Address: 211 Dacian Avenue, Durham, NC 27701. Internet: bhayes@amsci.org*

The Logic of Life

A digital technology usually starts with Boolean logic gates—devices that operate on signals with two possible values, such as *true* and *false*, 1 and 0. An AND gate has two or more inputs and one output; the output is *true* only if all the inputs are *true*. An OR gate is similar except that the output is *true* if any of the inputs are true. The simplest of all gates is the NOT gate, which takes a single input signal and produces the opposite value as output: *true* becomes *false*, and *false* becomes *true*.

In electronic circuits, a NOT gate can be made from a single transistor, wired so that a high voltage at the input produces a low voltage at the output, and vice versa. When the gate switches between its two states, it does so abruptly, like a snap-action light switch. It is this sudden, nonlinear response that gives digital devices their resistance to noise and error. Because a gate is either fully on or totally off, a signal can pass through a long chain of gates without degradation.

Are there any biochemical equivalents to transistor gates? As a matter of fact, yes: There are hundreds of candidates. Perhaps the most interesting among them are the mechanisms of genetic control, which switch genes on and off.

The archetypal example of genetic regulation in bacteria is the *lac* operon of *E. coli*, first studied in the 1950s by Jacques Monod and François Jacob. The operon is a set of genes and regulatory sequences involved in the metabolism of certain complex sugars, including lactose. The bacterium's preferred nutrient is the simpler sugar glucose, but when glucose is scarce, the cell can make do by living on lactose. The enzymes for digesting lactose are manufactured in quantity only when they are needed—specifically when lactose is present and glucose is absent.

As in the expression of any genes, synthesis of the *lac* enzymes is a two-stage process. First the DNA is transcribed into messenger RNA by the enzyme RNA polymerase; then the messenger RNA is translated into protein by ribosomes. The process is controlled at the transcription stage. Before the genes can be transcribed, RNA polymerase must bind to the DNA at a special site called a promoter, which is just "upstream" of the genes; then the polymerase must travel along one strand of the double helix, reading off the sequence of nucleotide bases and assembling a complementary strand of messenger RNA. One mechanism of control prevents transcription by physically blocking the progress of the RNA polymerase molecule. The blocking is done by the *lac* repressor protein, which binds to the DNA downstream of the promoter region and stands in the way of the polymerase.

When lactose enters the bacterial cell, the *lac* operon is released from this restraint. A metabolite of lactose binds to the *lac* repressor, changing the protein's shape and thereby causing it to loosen its grip on the DNA. As the repressor protein drifts away, the polymerase is free to march along the strand and transcribe the operon.

The repressor system is only half of the *lac* control strategy. Even in the presence of lactose, the *lac* enzymes are synthesized only in trace amounts if glucose is also available in the cell. The reason, it turns out, is that the *lac* promoter site is a feeble one, which does a poor job of attracting and holding RNA polymerase. To work effectively, the promoter requires an auxiliary molecule called an activator protein, which clamps onto the DNA and makes it more receptive. Glucose causes the activator to fall away from the DNA just as lactose causes the repressor to let go—but the ultimate effect is the opposite. Without the activator, the *lac* operon lies dormant.

All these tangled interactions of activators and repressors can be simplified by viewing the control elements of the operon as a logic gate. The inputs to the gate are the concentrations of lactose and glucose in the cell's environment. The output of the gate is the production rate of the three *lac* enzymes. The gate computes the logical function: (lactose AND (NOT glucose)).

A question remains: Do these biochemical control mechanisms exhibit the on-off, all-or-nothing character of digital circuits? Although the transition between states is never perfectly sharp, the digital approximation is often a good one. A factor that tends to steepen the response curve is the cooperative action of multiple subunits in the regulatory proteins. The *lac* repressor consists of four subunits, and the *lac* activator has two. Although the first subunit may be slow in binding to the DNA, subsequent units stick to one another as well as to the DNA, and so the binding goes faster. The net effect is to make the threshold for repression or activation sharper.

Biochemical Circuits and Networks

The analogy between metabolic regulators and digital logic was already noticed 40 years ago. In 1961 Monod and Jacob wrote about genetic circuits and switching networks, and they described how activator and repressor proteins could be organized into systems that would function as memory elements and oscillators. Other authors soon explored the connection between molecular biology and digital computing in greater depth and detail; indeed, for several years the theme was a frequent one in the *Journal of Theoretical Biology* and the *Bulletin of Mathematical Biophysics*.

The main focus of these early studies was on using digital models as a way of understanding events in the living cell. The Boolean approximation was a way of avoiding an unwieldy analysis of a complex chemical web. To follow all those molecular interactions in complete detail would have required tracking the concentrations of innumerable molecular species, measuring the rates of chemical reactions, and solving hundreds of coupled differential equations. Pretending that every gene is either on or off reduced the problem to a simpler digital abstraction.

The biological computer turns this idea upside down. Instead of constructing a computational model of biochemistry, you exploit quasi-Boolean biochemistry to do computing. This notion also has a history. In the 1970s Otto Rössler analyzed

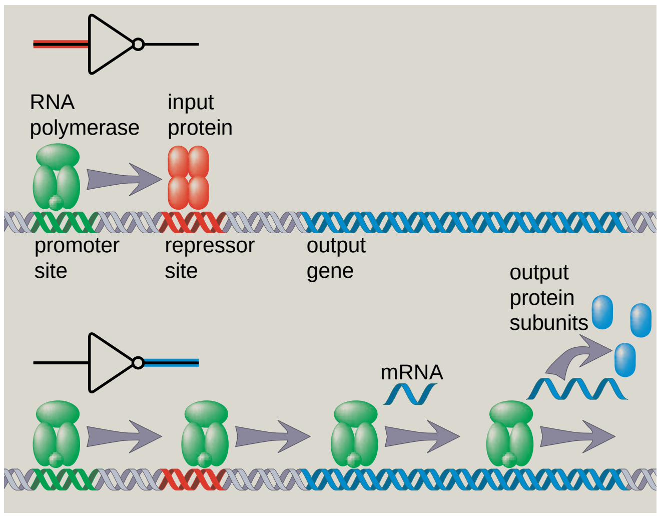

Figure 1. Biological NOT gate is implemented by a repressor-regulated gene. Presence of the input repressor *(top)* **blocks transcription of the gene; absence of the repressor** *(bottom)* **allows RNA polymerase to make messenger RNA for the output protein.**

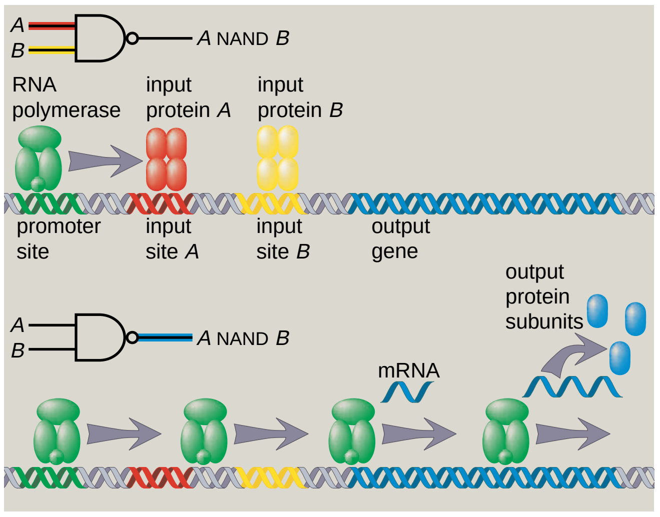

Figure 2. NAND gate is an elaboration of the NOT gate, fitted with two repressor sites in series. Either *A* **or** *B* **blocks transcription** *(top)*; **the output protein appears only when both** *A* **and** *B* **are absent** *(bottom)*.

various coupled systems of chemical reactions that could implement the abstract computers called finite automata. More recently, other groups have looked at schemes of computing based on the catalytic activities of enzymes.

The most novel plan for biologically inspired computing was conceived by Leonard M. Adleman of the University of Southern California. His basic idea is to use the complementary base-pairing of DNA as a pattern-matching engine. Adleman himself and others have demonstrated the feasibility of this idea in experiments where vials of DNA carry out computational tasks in number theory and combinatorics.

Bioware

All of the molecular computing methods mentioned above envision that the computation will be done *in vitro*. Although the molecules are of biological origin, they are extracted from the cell, and the reaction takes place in laboratory glassware. But why not turn the living cell itself into a computer, powered by its own metabolism? Several research collaborations have done work pointing toward this possibility. Here I shall focus mainly on the ideas of a group at MIT, who have examined the computational aspects of the problem in great detail. The MIT group consists of Thomas F. Knight, Jr., Harold Abelson and Gerald Jay Sussman, and several of their present and former students, including Don Allen, Daniel Coore, Chris Hanson, George E. Homsy, Radhika Nagpal, Erik Rauch and Ron Weiss.

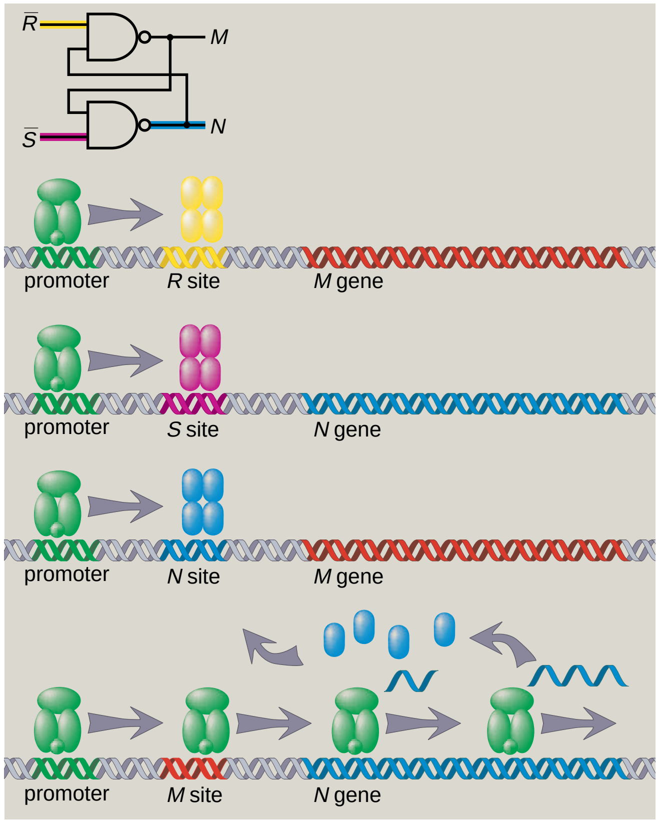

Figure 3. Biochemical flip-flop relies on cross-coupled genes, which inhibit each other's output. Here transcription of the *N* gene at the bottom shuts down the *M* gene just above it. Genes regulated by *S* and *R* ("set" and "reset") control the system: Momentarily removing *R* would trigger a switch to production of *M* instead of *N*.

The first major goal of the MIT group is to develop design rules and a parts catalogue for biological computers, like the comparable tools that facilitate design of electronic integrated circuits. An engineer planning the layout of a silicon chip does not have to define the geometry of each transistor individually; those details are specified in a library of functional units, so that the designer can think in terms of higher-level abstractions such as logic gates and registers. A similar design discipline will be needed before biocomputing can become practical.

The elements of the MIT biocomputing design library will be repressor proteins. The logic "family" might be named RRL, for repressor-repressor logic, in analogy with the long-established TTL, which stands for transistor-transistor logic. The basic NOT gate in RRL will be a gene encoding some repressor protein (call it *Y*), with transcription of the *Y* gene regulated in turn by a different repressor (call it *X*). Thus whenever *X* is present in the cell, it binds near the promoter site for *Y* and blocks the progress of RNA polymerase. When *X* is absent, transcription of *Y* proceeds normally. Because the *Y* protein is itself a repressor, it can serve as the input to some other logic gate, controlling the production of yet another repressor protein, say *Z*. In this way gates can be linked together in a chain or cascade.

Going beyond the NOT gate to other logical operations calls for just a little more complexity. Inserting binding sites for two repressor proteins (*A* and *B*) upstream of a gene for protein *C* creates a NAND gate, which computes the logical function NOT AND. With the dual repressor sites in place, the *C* gene is transcribed only if both *A* and *B* are absent from the cell; if either one of them should rise above a threshold level, production of *C* stops. It is a well-known result in mathematical logic that with enough NAND and NOT gates, you can generate any Boolean function you please. For example, the function (*A* OR *B*) is equivalent to (NOT (*A* NAND *B*)), while (*A* AND *B*) is ((NOT *A*) NAND (NOT *B*)). The NOT gate itself can be viewed as just a degenerate NAND with only one input. Thus with no more resources than a bunch of NAND gates, you can build any logical network.

Pairs of NAND gates can also be coupled together to form the computer memory element known as a flip-flop, or latch. Implementing this concept in RRL calls for two copies of the genes coding for two repressor proteins, *M* and *N*. One copy of the *M* gene is controlled by a different repressor, *R*, and likewise one copy of the *N* gene is regulated by repressor *S*. The tricky part comes in the control arrangements for the second pair of genes: Here the repressor of *M* is protein *N*, and symmetrically the repressor of *N* is *M*. In other words, each of these proteins inhibits the other's synthesis. Here's how the flip-flop works. Suppose ini-

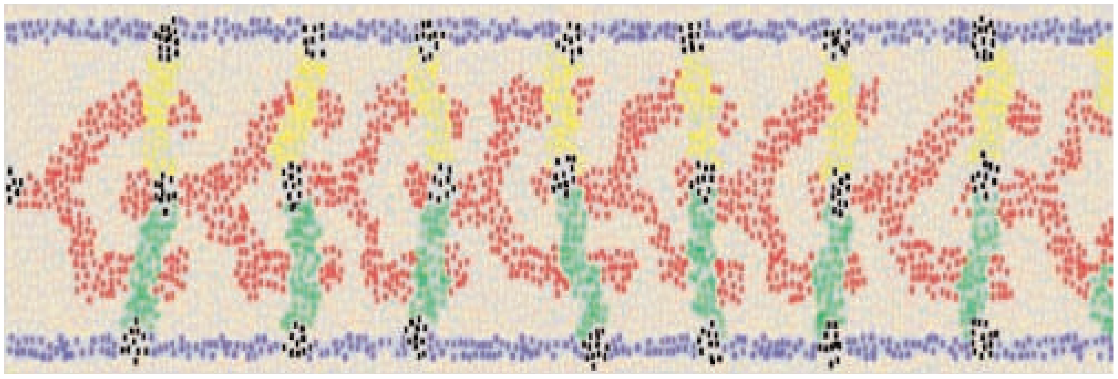

Figure 4. Multicellular biocomputer might organize its own structures much as tissues and organs differentiate. In this simulation an irregular and unsynchronized array of cells, all running the same genetic program, spontaneously generate a complex pattern—as it happens, the pattern for a chain of electronic NOT gates. (Image reproduced with permission of the MIT Artificial Intelligence Laboratory.)

tially that both *R* and *S* are present in the cell, shutting down both of the genes in the first pair; but protein *M* is being made at high levels by the *M* gene in the second pair. Through the cross-coupling of the second pair, *M* suppresses the output of *N*, with the collateral result that *M*'s own repressor site remains vacant, so that production of *M* can continue. But now imagine that the *S* protein momentarily falls below threshold. This event briefly lifts the repression of the *N* gene in the first pair. The resulting pulse of *N* protein represses the *M* gene in the second pair, lowering the concentration of protein *M*, which allows a little more *N* to be manufactured by the second *N* gene, which further inhibits the second *M* gene, and so on. Thus a momentary change in *S* switches the system from steady production of *M* to steady production of *N*. Likewise a brief blip in *R* would switch it back again. (*S* and *R* stand for "set" and "reset.")

One conclusion to be drawn from this synopsis of a few RRL devices is that a computer based on genetic circuits will need a sizable repertory of different repressor proteins. (I've used up a third of the alphabet already.) Each logic gate inside a cell must have a distinct repressor assigned to it, or else the gates would interfere with one another. In this respect a biomolecular computer is very different from an electronic one, where all signals are carried by the same medium—an electric current. The reason for the difference is that electronic signals are steered by the pattern of conductors on the surface of the chip, so that they reach only their intended target. The biological computer is a wireless device, where signals are broadcast throughout the cell. The need to find a separate repressor for every signal complicates the designer's task, but there is also a compensating benefit. On electronic chips, communication pathways claim a major share of the real estate. In a biochemical computer, communication comes for free.

Are there enough repressor proteins available to create useful computational machinery? Note that interference between logic gates is not the only potential problem; the repressor molecules taking part in the computation must also be distinct from those involved in the normal metabolism of the cell. Otherwise, a physiological upset could lead to a wrong answer; or, conversely, a computation might well poison the cell in which it is running. A toxic instruction might actually be useful—any multitasking computer must occasionally "kill" a process—but unintended events of this kind would be a debugging nightmare. You can't just reboot a dead bacterium.

Nature faces the same problem: A multitude of metabolic pathways have to be kept under control without unwanted crosstalk. As a result, cells have evolved thousands of distinct regulatory proteins. Moreover, the biocomputing engineer will be able to mix and match among molecules and binding sites that may never occur together in the natural world. The aim of the RRL design rules is to identify a set of genes and proteins that can be encapsulated as black-box components, to be plugged in as needed without any thought about conflicts.

Another important design tool is a simulator, which allows a device to be tested without the substantial effort of building a prototype. The world of electronics has long relied on a simulator called Spice, which models the physics of transistors and other electronic components. The MIT group is building a BioSpice simulator, which will model the dynamics of genetic circuits in a similar way.

So far, the MIT group has based their design work primarily on such simulations, but other groups have begun a few "wet" experiments. Michael B. Elowitz and Stanislas Leibler of Princeton University have created a free-running genetic oscillator in *E. coli.* Arranging three repressor genes so that they act on one another in turn, they observed periodic fluctuations in gene expression, with a frequency independent of the cell's reproductive cycle. In another *E. coli* experiment, James J. Collins, Timothy S. Gardner and Charles R. Cantor of Boston University built a genetic toggle switch much like the flip-flop described above, with two cross-coupled promoters and repressors. They report "robust bistability." Their eventual aim is the construction of "genetic applets"—self-contained program modules that could be "downloaded" into organisms.

Be Fruitful and Multiply

Persuading a living cell to perform useful computations is quite a trick, and yet it's not all that's needed. The real goal is to get a billion cells working in concert on the same task.

From one point of view, mass producing bacteria is extraordinarily easy. You don't have to build a billion-dollar "fab line" to manufacture them; just supply warmth and nutrients, and the bacteria will take care of proliferation on their own. The hard part is organizing a population of cells so that they work toward some specified goal. Here again electronic and biological technologies diverge. On a silicon chip, every circuit element has an assigned place and function, but living cells are squishy and motile and not easily confined to a rigid grid. The MIT group therefore takes another cue from biology, and lets large-scale structures emerge from processes akin to natural development and differentiation.

In an embryo, cells of identical genetic endowment differentiate into distinct tissues and organs, and also generate patterns such as the stripes and spots of animal pelts. What is most intriguing about biological development is that all the cells begin with the same "program," and they organize themselves without any externally designated leader. It all seems to be done by means of short-range communication between neighboring cells and the diffusion of chemical signals over longer distances. These same mechanisms would also be available to a multicellular biocomputer.

The study of large arrays of simple processing elements is a classical topic in computer science, but for the most part the arrays have been geometrically regular, and the processors have operated in strict synchrony. The MIT group offers a new paradigm of "amorphous computing" by spatially irregular and unsynchronized arrays. If all the processors run the same program, and they have only local communication, what patterns can emerge in such an amorphous blob of computers? Some of the examples generated so far have a distinctly botanical look to them, and yet they also resemble the design drawings for a silicon integrated circuit.

Many further hurdles remain before biocomputing could become a practical technology. Input and output are problematic. Maybe the input device will be a pipette of pheromone, and fluorescent proteins could produce output signals, but the expressive possibilities of these facilities seem rather limited. Large-capacity long-term storage—a biological disk drive—is also lacking. And speed is a concern, even with the extraordinary level of parallelism implicit in the exponential growth of a bacterial colony. Silicon processors are running at a gigahertz, but the speed of genetic circuits is in the millihertz range. Even a billion bacteria are no match for a Pentium.

But surely it would be a mistake to think of the *E. coli* computer as a beige box that will sit on your desk running a prokaryotic version of Microsoft Windows. A more likely prospect is a crop of programmable biological sensors, actuators and messengers. One contemplated application of such organisms is the assembly of nanoscale structures; instead of replacing semiconductor circuits, the cells would fabricate them. Another possibility is the old fantasy of a microscopic robot that could enter the human body to repair diseased tissues or combat infections. If this daydream of an intravenous computer is ever to happen, success seems more likely with the tools of genetic engineering than with a soldering iron.

Or maybe not. Maybe in the end it's just foolishness to imagine that anything so intellectually demanding as computation could be imposed on a biological substrate. No living organism can be expected to engage in abstract reasoning and symbol manipulation while carrying on with the daily routine of ingestion, growth, excretion, sleep, procreation. Get a life!

Bibliography

Abelson, Harold, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Radhika Nagpal, Erik Rauch, Gerald Jay Sussman and Ron Weiss. 2000. Amorphous computing. *Communications of the ACM* 43(5):74–82.

Adleman, Leonard M. 1994. Molecular computation of solutions to combinatorial problems. *Science* 266:1021–1024.

Coore, Daniel N. 1999. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. Ph.D thesis, MIT Department of Electrical Engineering and Computer Science. http://www.swiss.ai.mit.edu/projects/amorphous/papers/coore-phdthesis.ps

Elowitz, Michael B., and Stanislas Leibler. 2000. A synthetic oscillatory network of transcriptional regulators. *Nature* 403:335–338.

Gardner, Timothy S., Charles R. Cantor and James J. Collins. 2000. Construction of a genetic toggle switch in *Escherichia coli*. *Nature* 403:339–342.

Knight, Thomas F., Jr., and Gerald Jay Sussman. 1998. Cellular gate technology. In *Unconventional Models of Computation*, ed. C. S. Calude, J. Casti and M. J. Dinneen, pp. 257–272. Singapore, New York: Springer.

McAdams, Harley H., and Adam Arkin. 2000. Towards a circuit engineering discipline. *Current Biology* 10(8):R318–R320.

Monod, J., and F. Jacob. 1961. General conclusions: Teleonomic mechanisms in cellular metabolism, growth and differentiation. *Cold Spring Harbor Symposia on Quantitative Biology: Cellular Regulatory Mechanisms* 26:389–401.

Rössler, Otto E. 1974. Chemical automata in homogeneous and reaction-diffusion kinetics. In *Physics and Mathematics of the Nervous System*, ed. M. Conrad, W. Güttinger and M. Dal Cin, pp. 399–418. Berlin: Springer-Verlag.

Weiss, Ron. 1999. Programming colonies of biological cells. Ph.D. thesis proposal, Massachusetts Institute of Technology. http://www.swiss.ai.mit.edu/~rweiss/bio-programming/phd-proposal.pdf

Weiss, Ron, George E. Homsy and Thomas F. Knight, Jr. 1999. Toward *in-vivo* digital circuits. DIMACS Workshop on Evolution as Computation. http://www.swiss.ai.mit.edu/~rweiss/bio-programming/dimacs99-evocomp.ps