Published in final edited form as:
Mol Biosyst. 2009 July ; 5(7): 675–685. doi:10.1039/b902484k.

Biocomputers: from test tubes to live cells

Yaakov Benenson
FAS Center for Systems Biology, Harvard University, 52 Oxford Street Cambridge MA 02138 USA, Phone 617-384-7791, kbenenson@cgr.harvard.edu

1. Introduction

Computation, in a technical sense, is a standardized process by which input data are processed according to prescribed rules (algorithm) and are converted into output data. While computers do not "know" where the data come from, normally they represent a particular reality. For example, wind shear detector system on an airplane detects this dangerous atmospheric condition (change in wind speed over a short distance) by feeding radar readouts into a sophisticated computer program that alerts the pilots and helps to correct the flight course. In this case the interpretation of the outside conditions is coupled to a control function. Current silicon-based computers, combined with electronic input and output peripherals, are very successful in both interpreting and controlling large "chunks" of reality. Yet there is one realm where we largely stay helpless and can neither truly understand what is going on, nor affect the course of events – our own organisms and other biological systems. It is true that huge steps have been made in understanding basic biological processes as well as disease-linked abnormalities in humans. It is also true that any medical treatment is an attempt to control a disease, and we are witnessing an ever increasing number of efficient drug-based and surgical interventions. Nevertheless, were we to apply approaches used in modern medical treatment to flight control in airplanes, those airplanes would never take off due to the lack of spatial and temporal resolution of their controllers. That is, only very rough parameters would be estimated (instead of minute changes in atmospheric conditions), and similarly the control will come much later than needed, and often non-specifically. Therefore, the idea to gather and process information from various parts of our bodies, perhaps even individual cells, and use these data to control biological processes in real time, averting disease-linked transformations, is very appealing. However it may feel extremely uncomfortable to let an army of sensors, micro- or nano-computers and such roam our bodies not the least because we are not built to support tiny silicon-based devices in our bloodstream. Still, something has to be introduced from the outside. This "something" should be compatible with our physiology, in other words, comprise man-made, engineered molecular and cellular systems. But what if limiting ourselves to these biocompatible building blocks will necessarily mean that these "biological computers" will be vastly inferior to modern silicon computers and their ability to examine and control complex systems in real time? What types of computation are possible with molecules and cells? And what are biological computers anyway? This review will try to examine the answers given to these questions by researchers in the field and practical examples of prototypes.

The idea that molecules can compute was proposed by a group of computer scientists and electrical engineers who observed the way information is processed in living organisms and cells, and compared it to what they knew about theoretical computer science and computer engineering[1-13]. It has long been observed that the abstract notion of "computation" can be implemented in a large number of ways, including such weird set-ups as a "billiard ball" computer[1]. As a more serious argument, the entire concept of "computation" was established as a formalization of the way humans think and process information, that is, the working of the brain[14, 15]. And while the brain can compute, and it does so using cells and chemicals, it

is less obvious if smaller-scale biological objects such as individual cells or even collections of molecules can compute, too.

On one hand, an affirmative answer to these questions was given with the discovery of biological regulation by Jacob and Monod[16-18]. It turned out that molecules connected into regulatory networks can convert regulatory molecular inputs into specific molecular responses (outputs), and they do so using certain input-output relationships or functions. For example the presence of lactose and the lack of glucose in the bacterium growth medium will induce the Lac operon and elevate the expression of genes LacA, LacY and LacZ. Roughly speaking, the network computes a logic function (Lactose AND NOT Glucose → LacA, LacY, LacZ)[19]. While the real relation is more complicated, the bottom line is clear: the network "computes" a function that connects the concentrations of lactose and glucose in the medium to the Lac operon activity. Therefore what we observe here can be called a biomolecular *computation*[20].

On the other hand, a deeper question about the lactose utilization network that has led to confusion is whether this network constitutes a biomolecular *computer*. The definition from the Oxford English Dictionary states that a computer is "an electronic device (or system of devices) which is used to store, manipulate, and communicate information, perform complex calculations, or control or regulate other devices or machines, and is capable of receiving information (data) and of processing it in accordance with variable procedural instructions (programs or software)." What is implicit in this definition is that a computer is a device that can do multiple things. This does not mean that all devices can multitask – the world is full of task-dedicated machines, such as cars or CD players. However, the ingenuity of the digital computer architecture is such that it can act both as a music player (i.e. iPod) and as a car simulator. On the contrary, a Lac network in bacteria can only do one thing, albeit exceedingly well. Evolutionary pressure[21] and random mutagenesis[22] were shown to alter the Lac network computation, and it would be surprising if it did not. However, in an individual cell the Lac network is not being reprogrammed in a way our personal computers are reprogrammed every time we use a web browser or a text editor. The question arises if and when to call a molecular interaction network a biomolecular computer.

To thoroughly present the way computers are built is beyond the scope of this review, but we can summarize a few intuitively understandable features. First, computers are given tasks that can be described in common language (such as, the task could be to "find an average value of the set of numbers". This task needs to be translated, firstly, into a high-level algorithm (*e. g.* "(1) add all numbers; (2) keep track of their total number; (3) divide one by another"). Secondly, the algorithm is implemented in a specific programming language. Finally, this program is translated into a machine language, usually by another program called a compiler. The program in the machine language, or *software*, is loaded into appropriate storage in the computer *hardware*. This is not in itself enough to implement the task. The next stage is to provide the input data, that is, the numbers we want to average. These numbers may come from any number of sources – temperature readouts, historical stock prices or results of a repeated experiment. The data from these sources have to be converted into machine representation. If we are averaging temperatures, we will use digital thermometers that generate a digital signal proportional to the actual temperature; we will record the readouts on some form of digital media, *e. g.* a CD. Finally we will insert the CD into the CD drive on our desktop computer, where the stored values will be uploaded into the computer memory. Please note that at none of these steps do we need to affect the software or the hardware in any way; these have been built completely independently of the actual data they are designed to process. Instead, we used an elaborate sequence of *peripheral input devices* whose task is to faithfully deliver the data from their origin to the computer (Figure 1A). Let us elaborate a little further and imagine that this average temperature is not only calculated to satisfy our curiosity, but is also used to control

something else. For example, the temperature in question could be collected from different locations in a chemical reactor; deviations of the average temperature from a required set-point should trigger cooling down or heating action. This can be accomplished by a small additional piece of software that calculates the difference between the average temperature and the set-point and opens either a cooling valve or a heating valve when the difference is respectively positive or negative. To do this, a digital signal generated by the computer and representing either “open cold” or “open hot” command needs to be fed into a device that actually moves the valves. This device is also peripheral to the computer and is a part of neither software nor hardware.

To summarize, what makes a computer such an outstanding machine? First, it possesses flexible hardware that can be programmed with any number of algorithms and store large amounts of data. The computer is oblivious to the source or nature of the data as long as the data are presented to it in a convenient digital format. Instead, the task of delivering the data and translating the computer's output into action is deferred to any number of input and output peripherals that are built separately in a modular fashion. A particular case of this arrangement is a setup where specific input and output peripherals are connected to a computer that runs only one, possibly very complicated, program. This system would be called an automaton, after Norbert Wiener; most robots, in particular advanced autonomous devices such as extraterrestrial probes, are automata in this sense. A more far-fetched analogy from a living world is a bacterium cell, with its receptors representing the input peripherals, its entire internal network representing the computer and the cell phenotypic state at a given moment representing the output[23]. However, this will not count as a biological automaton but only as “automation” (analogous to biological “computation” on a smaller scale), because the “computer” inside the bacterium is really a *computation* as it cannot be easily reprogrammed.

Following this discussion, we can formulate what constitutes a biomolecular computer: (1) the presence of the molecular “hardware”, the invariant part of the network that can support versatile algorithmic tasks; (2) the ability to convert well-defined tasks and algorithms into molecular “machine language” in a deterministic, automated way; (3) availability of special machine format for data representation; (4) availability of input peripherals that deliver molecular data from diverse sources and convert them into the molecular “machine format” for algorithmic processing; and (5) availability of output peripherals that convert data from the machine format into diverse biological responses (Figure 1B). Significant progress has been made in implementing these requirements in both biochemical and biological realms, and this progress is reviewed below. It should be noted that the requirements from an ideal biocomputer are very broad and it is not clear whether a system that implements them all is at all feasible. Instead, we consider different systems and ask which requirements they do fulfill, and to what extent.

2. DNA-based *in vitro* biocomputers

DNA has traditionally been a favorite building block for molecular computations and biocomputers. However, while DNA is a biological molecule, in nature it normally serves to store genetic information and less as an active participant of reaction networks. Therefore, DNA-based systems have been mainly implemented in test tubes where well-designed species have been assembled and their emergent computational behavior has been observed. Although not biological computers in the strict sense, DNA systems inspired and informed a great deal of experimentation in biological systems. They also afforded in-depth exploration of molecular computations unhindered by the sheer technical challenge of working with living organisms.

2.1. Tiling systems

One of the earliest proposals for a general-purpose biocomputer was made by Eric Winfree and colleagues[24]. He examined a model of computation called “tiling”. Without entering into details, tiling is a way to simulate a Turing machine by sequential self-assembly of so-called “Wang tiles”. In the Turing machine description[14], a tape of symbols is modified one symbol at a time by a programmed controller that can switch between a finite number of states. The controller can read the symbols from the tape and, depending on the readout and on its own state, it can write a new symbol, change its state and move one symbol to the right or to the left. The combination of these rules constitutes the Turing machine's program. At each computational step only one symbol is changed in the entire tape, together with the controller state and position. This instantaneous combination of the tape and the controller state and position is called a configuration. In the tiling approach, each configuration is represented by a row of tiles, with most tiles containing the tape symbols and one tile containing the controller state. (The location of the controller tile within a row indicates which symbol it is pointing at.) In order to move to the next configuration, a new row of tiles is assembled on top of the original row. The rules of assembly are encoded in the tiles' edges, telling them which tiles they can bind to. Properly designed tiles can encode the entire program of the Turing machine. Given the initial configuration of the computation (the first row), simply adding enough tiles to the system and letting them assemble according to their edges' features will automatically generate all subsequent computational configurations till completion. A by-product of the process will be a plane filled with tiles encoding the entire history of the process. If the internal areas of the tiles (as opposed to edges) contain interesting information, this history can result in intricate, a-periodic patterns.

Winfree proposed that rigid DNA structures called “DNA tiles”, built earlier by Nadrian Seeman and his lab members[25], could implement the tiling model. In his proposal, the DNA tiles were adorned with specific single stranded “sticky ends” that effectively represented the edges of the theoretical Wang tiles. Thrown in solution, the DNA tiles would physically self-assemble and as the argument went, realize the tiling model by actual creation of a flat DNA surface where each tile's location is determined by the computational algorithm encoded in the tiles' structure. While the theory behind DNA tiling has been formulated a while ago, its experimental demonstration had to overcome numerous obstacles, mainly due to erroneous incorporation of tiles. However, significant progress has been made[26, 27], and remarkably this approach spurred the advent of the “DNA origami” technology[28], which, while not being Turing-equivalent, enables rapid and robust construction of nano-scale objects with complex long-range features.

Considering the tiling approach in the light of the “biocomputer” definitions above, its strengths are in the hardware and software components. In principle, any computation can be encoded with a finite number of different tiles, and given their reliable assembly the correct output will be produced. One of the first logic computations using DNA tiles was demonstrated by Seeman and Reif[29], where a two-row assembly correctly calculated a nested logic function of four inputs. It is less clear how to couple the system to peripheral inputs and outputs. The input data in the machine format has to be implemented as a set of tiles, and these tiles have to be carefully assembled to initiate the process. Conversion of input data in other formats into this configuration will have to be addressed in the future, as well as translating the machine output into desired responses. In many cases the actual machine output, that is, 2D DNA surface is the desired output that can be used for nanotechnology applications.

2.2. State machine systems

Like tiling, state machine paradigm was inspired by the Turing machine formalism and by the perceived similarities between its data tape and DNA strands. Unlike tiling, this paradigm

sought a more direct implementation of Turing's ideas, that is, physically building the tape, the controller, *et cetera*. Theoretical approaches were proposed early on, and their common features were (1) encoding the machine's configuration, *i.e.* the tape and the controller, in a specially designed DNA molecule with certain sequences representing different symbols and others representing states; (2) programmed alterations to this DNA tape using a series of biochemical transformations, effectively performing one computational step at a time. Two notable blueprints were proposed by Smith[30] and Rothemund[31]. Shapiro took a different approach, showing a ribosome-inspired blueprint of a molecular Turing machine that combined some ideas from "tiling" with reactive transformations[13, 32], with an emphasis on input and output peripherals and the operation of this machine as an autonomous, reactive entity in a biological host.

While Turing machine is a universal computer and a molecular Turing machine would be a perfect biocomputer, none of the aforementioned and additional proposals have been implemented so far. Instead, researchers looked into more limited versions of Turing machines, cumulatively known as finite state machines or finite automata. In general, their limitations are derived from the way they are programmed. The program for a Turing machine is a collection of rules of the form <current state>, <current symbol> → <next state>, <next symbol>, <move Left>/<move Right>. Increasingly simplified state machines reduce this general form, for example, by only allowing the controller to move right (state, symbol → state, symbol, Right), or adding on top of that the inability to write new symbols (state, symbol → state, Right), or removing the symbols altogether (state → state).

The first experimental state machines built by Hagiya, Sakamoto and colleagues[33, 34] were of this latter simplified flavor. The system was physically implemented by cycles of DNA molecule extensions using DNA polymerase. The program or the transition table of the computer, *i.e.* the set of <current state> → <next state> instructions, was encoded in a single-stranded DNA molecule; the initial state of the system was a segment of DNA physically attached to the "transition table" molecule. The initial state segment would then iteratively hybridize to its complementary fragments in the transition table moiety, and get extended by DNA polymerase into the next state segment, and so on. In theory, the process may proceed indefinitely, resulting in an algorithmically-defined DNA strand. In practice mis-hybridizations and end-product inhibition limited the number of extension steps, although impressive progress has been made and ten transitions shown[34]. Judged against biocomputer criteria, this state transition device was in theory capable to implement any finite set of state transition rules. However, the "software" and the input components of the computation had to be physically constructed before the computation could commence. Accordingly, there were no obvious ways to provide the system with input peripherals. On the other hand, the output of the process, *i.e.* the long DNA molecule encoding the history of state transitions, could naturally feed into other biochemical processes for example via reverse transcription into mRNA and translation into protein, or by PCR amplification to generate a functional gene.

Extensive work on state machines has been performed by the author, Ehud Shapiro and colleagues[32]. Transitioning thorough multiple reincarnations[35-37], we arrived at a molecular two-state finite automaton that could run "if then else" decision-making algorithms[38]. The algorithms we could program in our system were of the form

Begin

Assign initial state to Yes.

Iterate through conditions

If current condition holds and current state is Yes, remain in state Yes else switch to No

If current state is No stay in state No

End.

On the molecular level we used a specially designed "computational" DNA molecule to implement the algorithm (Figure 2A). It has a number of rationally-designed double-stranded segments, each representing one condition and one algorithmic step. The leftmost segment has a label – a few of its nucleotides exposed as a single-stranded overhang – indicating that the current state of the computation is Yes, with the segment itself representing the *next condition to be checked*. Elsewhere we introduced a mechanism for condition testing (see below), that we can consider for now as a black box (Figure 2A). The job of this box is to make two kinds of molecules. If the condition holds, the box makes a "positive transition molecule"; if not, a negative transition molecule. These molecules are designed to recognize the state labels on the computational DNA; each "condition segment" requires a unique pair of molecules because its sequence and hence the sequence of the state label are also unique. Transition molecules become cofactors of a restriction enzyme FokI and direct the cleavage of the computational DNA in such a way that the leftmost segment is chopped off and the next segment becomes exposed and tagged with the state label. A positive transition preserves the state; a negative transition generates a different label representing the No state (Figure 2C). The black box operates repeatedly until all the conditions have been checked. However, if at any time the state has switched to No, pre-made transition molecules that preserve the No state would unconditionally chop off the remaining segments one by one. A special state label generated after all the conditions have been processed amounts to the computational output and represents the decision; the final state is Yes and the decision is positive only if all the conditions hold at the same time. The system is oblivious to what the actual conditions are; as long as the black box works, it makes correct decisions. It is fairly flexible as more conditions can be checked by adding more segments into the "computational" DNA, and different sets of conditions can be checked in parallel by combining two different "computational" DNA molecules in the same mixture. Detailed analysis of the system showed that it is in principle capable of arbitrarily complex Boolean decision making[39].

Let us turn to what is going on in the black box (Figure 2B). We operate in the molecular world, and the presence of a condition would normally mean that a certain input molecule is present at a high concentration. Alternatively, it may mean that a certain molecule is absent, or that there is a difference such as mutation between the wild-type and the condition we are looking for. To implement the black box we needed a mechanism that would convert high concentration of the input molecule into high concentration of the positive transition, and low concentration into a negative transition. Condition signified by low concentration should trigger a positive transition when concentration is low, and a negative transition when it is high. Finally, for a mutation-linked condition a mutated molecule should activate a positive, and a wild-type a negative transition. While these requirements are very generic, we implemented them only for one family of molecular species, namely messenger RNA molecules or transcripts. In nature these molecules mediate the synthesis of proteins from genes, and their concentrations correlate fairly well with the levels of proteins they code for and consequently with the cell phenotype. Let us consider an example of a high concentration condition. The trick was to initialize the system to make a default switch into a negative state if the condition did not hold, or if the concentration was low, due to the *a priori* requirement to make a default negative decision. Therefore we already included a "waiting" negative transition molecule in the mixture. At the same time we also included an inactive form of a positive transition. The "black box" has to implement molecular pathways that, in the presence of high mRNA concentration, would activate the previously inactive positive transition and inactivate a previously active negative transition. As it turns out, there are no extra components in the back box – all that needed was an appropriate design on the transition molecules, based on the nucleotide sequence of the

mRNA input signal. Similar mechanisms work for low-concentration and mutation conditions, and they occur for each iteration step.

As stated above, mere decision making is not enough, because a decision has to be translated into action. In daily life a decision-making computer program can give recommendations to be implemented by a person or another machine. However, in the molecular world a decision has to be tightly linked to action, because it may be impossible to read out the decisions from individual cells. In our case we added a DNA hairpin modifier to the right-hand tail of the computational DNA molecule. The loop at the hairpin's end is an antisense DNA that can regulate gene expression. After all the conditions have been processed, the DNA molecule is labeled with either a Yes or a No final state. Separately engineered dummy transition molecules chopped away on the double-stranded segment of the hairpin; only in the final Yes state the dummy transitions would recognize the state label and remove the stem, exposing the antisense DNA fragment.

2.3. Logic networks

Separately from Turing machine-inspired research, a parallel effort explored molecular computation paradigms inspired by “logic circuit” architecture. In fact, logic circuits are the workhorse of silicon computers. Researchers who pursued logic circuit ideas drew parallels between the passage and alterations of voltages in electronic circuits with changing concentrations of molecular species in networks of coupled chemical reactions. For example, if two chemicals A and B were simultaneously required to catalyze the production of a chemical C, this would be interpreted as an “AND” logic gate between the inputs A and B with an output C. In the molecular circuit paradigm, the importance of digital information stored in DNA sequence is greatly diminished compared to the state machine-based approaches. Instead, an entire molecular species is either in state Off (*i.e.* low concentration) or On (*i.e.* high concentration). An inherent difficulty with both interpreting and engineering digital reaction networks lies in the arbitrary definition of Off and On states. Clearly, concentrations can take any value between zero and infinity, and while defining the Off state is easy, doing so for the On state has to be justified. Fortunately, in the biological world the On states can be compared to what is known as “saturating concentrations”, that is, levels beyond which the effects exerted by a molecule stops being concentration-dependent. This is due to the fact that almost all natural processes are catalytic in nature and easily lend themselves to saturation.

Simple logic gates made of organic molecules predated biomolecular-based systems[40]. However, those gates often had different input and output formats, for example ion inputs and fluorescent outputs, and could not be easily integrated into circuits and cascades. First examples of biocompatible, scalable systems were described by Stojanovic, Stefanovic and colleagues[41]. In their works, so-called allosteric ribozymes[42-46] were controlled by multiple oligonucleotide inputs. Careful design of the ribozymes ensured for example that one input had to be present while at the same time another had to be absent, *i.e.* implementing an AND-NOT gate[47, 48]. The power of their approach was demonstrated by building a molecular automaton that could play and win the game of “tic-tac-toe” against a human opponent on a 3×3 grid[49, 50]. Molecular computations involved in the game involved a large number of mutually exclusive parallel logic operations. Moreover, the systems were not limited to single layer-networks, and the outputs of the logic gates – short oligonucleotides – were shown to act as inputs for downstream gates with the help of ligation reactions[51]. In a separate work, Levy and Ellington also showed that products of ribozyme-catalyzed reactions can trigger downstream processes[52], showing that ribozyme-based systems can lead to circuits of both substantial width and depth. In summary, ribozyme-based approach is very promising because these networks can be programmed to compute complex logic functions. One component that has yet to be shown is the input peripherals. The “machine format” of these circuits are DNA

or RNA oligonucleotides, and their levels need to be affected by a broad range of biologically-relevant inputs to make the system able to function in flexible biological context.

A more recent line of work from Winfree and colleagues[53] built on "nucleic acid devices"[54-57] and exploited them to build large-scale DNA logic circuits. Those devices were originally conceived to transduce molecular DNA signals into mechanical action by means of strand hybridization, strand migration and subsequent conformation changes. However, as in the decision-making state machine, strand migration processes can be exploited to initiate reaction cascades. Moreover, both inputs and outputs of nucleic acid devices are DNA oligonucleotides, and hence they naturally lend themselves to cascading. Winfree and colleagues were able to convert these devices into logic primitives by making the generation of the output conditional on two or more inputs. In addition, they could program complex signal-transduction functions, most notably signal restoration elements that are crucial for large-scale networks. Their approach can lend itself to complex logic circuits, both in theory and in practice. As with previously-described ribozyme-based networks, a remaining issue is the input peripherals that will transduce biological inputs into internal molecular format of short DNA or RNA oligonucleotides.

3. Protein-based *in vitro* systems

Proteins have been explored in the molecular computation context *in vitro*, both as enzymes and as regulatory motifs. One of the early attempts to utilize enzymes was presented by Sivan and Lotan[58, 59]. In that work, an enzyme chymotrypsin was chemically modified to render its activity sensitive to certain irradiation wavelengths (with one wavelength capable of activation and another of inactivation). In addition, they employed a small molecule inhibitor that could be rendered inactive by a reducing chemical environment. Taken together, only the combination of active enzyme form and inactive inhibitor would trigger an output from the gate. In a related line of research, purely molecular enzymatic systems were developed by Willner and colleagues[60-62], and later by Katz and colleagues[63]. These systems showed complex logic integration of molecular inputs as well as cascades of gates. Another example of protein-based system was shown by Libchaber *et al*, who interpreted the binding of RecA protein to DNA substrate as a stochastic Turing machine computation[64].

Peptides were proposed as building block for logic gates[65], serving as catalytic templates for condensation of other peptides from partial-length precursors. On a chemical-network level, the AND gate was implemented by using two different peptide templates catalyzing the same condensation. The NOR gate was implemented by inhibiting an autocatalytic condensation process independently by two other peptide inputs. Protein-based networks were also implemented in cell-free extracts, where biological processes of transcriptional regulations were reconstituted[66]. This direction is promising for "lab-on-the-chip" applications as well as a way to quantitatively test circuits whose ultimate goal is to operate in cells.

Enzyme-based circuits have an advantage of being inherently biocompatible. However, it has yet to be shown that these circuits can enable complex programming characteristic of DNA biocomputers. One promising application of these circuits is their utilization as components of larger networks under appropriate circumstances. In addition, developments in enzyme engineering could enable enzyme manufacturing with pre-designed function.

4. *In vivo* biocomputers

4.1. Introduction

The development of *in vivo* computational networks has mirrored the *in vitro* efforts in many aspects, albeit until recently without much cross-fertilization between the two fields, viewed

respectively as branches of synthetic biology and of DNA-based computing. While DNA-based networks have relied heavily on the primary DNA sequence as information carrier, *in vivo* systems adapted existing mechanisms for biological regulation, in particular transcriptional and post-transcriptional regulatory links, and generally adhered to logic circuits as the guiding model of computation. In a nutshell, most biological regulation interactions can be classified as either activating or inhibitory. Moreover, most of them are subject to saturation (see above). Therefore, if element A activates element B, and element A is saturated with respect to this interaction, we can define this process as a transmission of one bit of information through the network. If A inactivates B, then similarly the bit encoded in A is negated (inverted). More complex patterns will arise when elements A and B are both needed to activate element C (in which case C will carry the value of Boolean A AND B), or either A or B separately inactivate C (C = NOT(A) AND NOT(B) = A NOR B). Conceptually these ideas are easily comprehensible and their theoretical presentation dates back to Jacob and Monod, and to a series of papers by Sugita[67-71], and later by Hjelmfelt, Arkin and Ross[72-75]. However, practical implementation of novel engineered logic networks in cells with specified properties has encountered enormous difficulties that have only recently become partially resolved. The current state of research in these areas will be discussed below.

4.2. Protein-based systems

Chronologically, protein-based regulation of gene expression by transcription factors was the first regulation mechanism used for *in vivo* biological computers. Transcription regulation has been intensely studied in the past decades, and a wealth of experimental data has revealed how natural circuits operate. In some cases, computational models have been built that predict how various transcription inputs regulate complex promoters[76, 77]. However, these methods have generally relied on previously-gathered experimental data to fit parameter values or implement machine learning. Predicting gene regulatory function from first principles still remains a major challenge. Moreover, even if such tools existed, they would not be able to tell us how to connect arbitrary transcription factor inputs to a specified output gene in a desired fashion – something we would expect from a programmable system. Engineering specified regulatory behaviors with arbitrary transcriptional regulators is therefore one of the main unsolved issues in synthetic biology and molecular computing.

Weiss, Homsy and Knight showed that at least theoretically, gene regulation can form the basis for building NAND and NOR gates[78]. Both types of gates are universal and circuits of sufficient number of such gates could in principle lead to arbitrarily complex computational networks. Besides, theoretical constructs were proposed to compute logic expressions using transcriptional regulation in so-called normal forms[79] (Figure 3A). Therefore it is somewhat surprising that despite solid theoretical foundations and intellectual maturity, actual implementation of large-scale transcriptional logic networks has been limited. Granted, many exciting synthetic networks with interesting properties relied solely on cascades of inverters or activators, or included negative and positive feedbacks[80-88]. Promising results on implementing transcriptional logic in mammalian cells was shown by Fussenegger and colleagues[89]. More recently a modification of yeast tri-hybrid system was shown to implement complex logic using small molecules as inputs and a set of interacting proteins as mediators[90]; a two-input AND gate using a complex promoter has been shown in bacteria[91]. It is hard to tell whether the scarcity of reports is due to lack of trying or to experimental challenges. For one thing, the available repertoire of transcription factors readily available for incorporation in synthetic networks is small, and includes such well-known proteins as rtTA and LacI. Large networks will require assembly of tens of factors that need to be thoroughly characterized. There are also generic challenges pertaining to putting together and testing tens of gene components in a biological context.

The future potential of large transcription-based logic networks will depend on addressing a number of challenges. Using a "universal gate" approach will require extensive cascading of gates with protein outputs of upstream gates serving as inputs to downstream gates. There are intrinsic time delays for propagating the signal through multiple layers, as each stage requires both transcription and translation of protein[92]. Characteristic times required to accomplish this task depend on the host organisms, and non-surprisingly they are connected to other time scales of the hosts, in particular generation time. In applications where time is an issue, using more than three-four layers will make the network compute longer than it takes for the host cell to divide, leading to loss of resolution. Signal dissipation due to biological "noise"[93] in deep cascades is another issue to consider. It is also interesting to note that nature's own transcriptional "computations" are rarely deeper than 2-3 layers, reflecting similar constraints[94]. An alternative to deep cascades are wide and shallow circuits. Logic "normal forms" naturally lend themselves to such circuit architecture, but theoretical proposals to build such circuits[79] have not been implemented yet. All the above challenges, however difficult, are well worth solving because transcription factors can be coupled to a large number of input peripherals. They can be directly affected by small molecule metabolites or exogenous effectors, be themselves expressed from genes controlled by known inputs, and so forth.

A different approach to making Boolean calculations in cells proposed by Lim and colleagues has exploited signal transduction pathways[95, 96]. They used a regulatory protein N-WASP that is itself regulated by two molecular effectors in an endogenous context. This regulation is achieved through modular protein domains that both need to bind their corresponding effectors to "unlock" the actuator domain of N-WASP, resulting in an AND-like behavior. Lim and colleagues successfully replaced the endogenous input domains and preserved the AND character of the switch. Reengineering and co-opting signaling elements in molecular computational networks is very attractive because the time scales of these processes are typically much shorter than those of other biological regulation mechanisms. However, as the aforementioned work shows, we are still far from achieving true modularity with these elements and from building large-scale circuits. A possible solution may come from the two-component signaling pathways in bacteria that have been shown to be rather modular and reprogrammable, at least as far as individual proteins are concerned[97].

Finally, in a series of reports Chin and colleagues described synthetically modified ribosomes that could enable *in vivo* logic operations[98, 99], opening another exciting avenue for biological circuit engineers.

4.3. RNA-based systems

RNA has served as a bridge between the *in vitro* and *in vivo* networks. RNA was used in "traditional" DNA computing experiments[9], and it is increasingly being realized that the combination of RNA information-storage capacity and the fact that RNA can be synthesized in cells make it an ideal substrate for *in vivo* biocomputers[100, 101]. Moreover, a number of recently-discovered natural regulatory mechanisms that involve RNA arguably make it one of the most versatile compounds in circuit engineer's hands.

The regulatory mechanisms that involve RNA are multiple, and they have been described in a number of excellent reviews[100, 101]. Briefly, there are two broad categories: riboswitches and small RNAs in RNA interference (RNAi) pathway. Natural riboswitches are normally a part of mRNA transcripts and they form locally stable structures (such as stem-loops) in their "ground" state[102, 103]. The ground state can either enable protein translation from that mRNA, or inhibit it. The conformation of the switches can be altered by a number of inputs, such as temperature, small molecules, or other RNA molecules[104]. Upon interaction with the input, the switch shifts into an "active" state, which changes the pattern of protein expression (from Off to On, or the other way around). This active state can be reversible, or trigger irreversible

self-cleavage upon becoming an active ribozyme. Importantly, individual riboswitches can be modulated by a number of inputs or multiple riboswitches can be incorporated in the same mRNA; both configurations and their combination can enable complex regulatory processes and can be exploited to engineer logic networks[105].

Riboswitches have been extensively studied for *in vivo* computational networks by Smolke and colleagues[109-111]. Their yeast-based networks involved modification of a reporter gene's mRNA to include a number of riboswitches that responded to small molecule inputs and implemented a number of two-input logic gates. The framework can conceivably be extended to receive more inputs, because the riboswitches are integrated in tandem and they can implement both proportional and inverse signal transduction (*i.e.* the ground state can either lead to low or high reporter gene expression). The issue of versatile input peripherals responding to molecules other than metabolites needs to be addressed to render a truly universal approach to *in vivo* computing, although endogenous metabolites as inputs to the circuit can cover a wide range of biologically-interesting states. In terms of programmability the networks show great promise as a basis for shallow, wide circuits implementing normal form-like computations.

Another RNA-based mechanism, RNAi is a constitutive regulatory modality in higher organisms[106]. The mechanism enables the cells to elicit mRNA-directed downregulation of gene expression. All that is required to direct RNAi against a gene is a small RNA molecule whose sequence is partially or fully complementary to a segment of about 20 nucleotides long in a mRNA transcript of this gene (usually in either the coding region or 3′-UTR). These small RNAs can come in a number of flavors. Small interfering RNA (siRNA) are synthetic RNA duplexes 20 base pairs (bp) long, and they are supplied exogenously to cells. Short hairpin RNAs (shRNA) are RNA hairpins with a stem of ~20 bp long and a loop of 5 to 20 nt. They can be added exogenously or expressed from DNA constructs inside cells. Finally, microRNA (miRNA) are natural ingredients of the RNAi pathway[107]. miRNA are formed in cells through multiple processing steps, starting with an RNA transcript that contains a characteristic hairpin motif and ending in a single RNA strand ~ 20 nt long incorporated in a protein RISC complex and ready to downregulate a target gene. Synthetic miRNA genes can also be constructed following nature's cues[108].

RNAi regulation was shown to support large-scale logic computations in normal forms by the author, Weiss and colleagues[112]. In our report we showed a blueprint for logic circuits that are built around normal logic forms. As with riboswitches, the programmability of the system relies on the fact that only a small portion of a mRNA (*i.e.* “targets”) is required to establish one inhibitory link between the small RNA and this mRNA. The targets can be introduced in arrays in the gene's 3′-UTR, independently on the coding region, implementing NOT-AND-NOT-AND-NOT… logic operations, and multiple mRNAs with identical coding region can be combined in the same network, implementing an OR-OR-OR… logic (Figure 3B). In our blueprint, the small RNAs are “machine representation” of actual inputs to the circuit. We envision input peripherals that convert biological signal to sRNA format in either proportional or inverse manner. In the former case, the absence (or “zero/False” value) of the signal will lead to output production, and this signal will be negated in the formula computed by the circuit. In the latter case, the presence (“one/True” value) of the signal will lead to output production and hence the signal will enter the formula directly (Figure 3C). While we showed that the computational module can process up to five siRNA inputs making it the largest *in vivo* computation to date, the possibility of building these peripherals has yet to be shown. In a promising series of reports, Yokobayashi and colleagues as well as Smolke and colleagues showed that small molecules could modulate the activity of specially designed shRNA constructs fused to aptamers[113-115].

4.4. Hybrid systems

While many systems are purely protein- or RNA-based, using hybrid networks that combine both types of elements is increasingly getting more traction. A combination of protein and RNA regulation to control gene expression and in complex synthetic networks has been shown in a number of reports and often found to be superior to either mechanism. In the context of computational networks, these possibilities are only beginning to be exploited. A recent work by Voigt and colleagues constructed a gate based on transcriptional and tRNA inputs[116]. Our own work on RNAi-based circuits utilized a combination of RNA and transcriptional regulation to enable computations in CNF logic form[112]. I believe that ultimately, the large-scale logic integration will be deferred to RNA-based regulation, while specific elements of the networks will judiciously employ transcriptional, signaling and enzymatic elements.

5. Conclusions

I have provided here a brief survey of the emerging field of molecular and biological computation. The field rests on ideas and methods developed in the areas of DNA- and RNA-based computing on one hand, and synthetic biology on the other. However, this is a stand-alone effort as both its intellectual predecessors are much wider in scope (in particular, synthetic biology that includes just about everything to do with rational design of biological systems). Biocomputers are conceptually well-defined species whose practical implementation is very challenging. The initial challenge of simply thinking about the possibilities and formulating the right questions is probably behind us. The field seems to have identified a number of tractable goals as well as long-term objectives. In the immediate future we have to show that *in vivo* computations using transcriptional, post-transcriptional and post-translational networks and the combination of thereof can move beyond proofs-of-concept and simple circuits with a small number of elements to much more complex systems that can solve real-life problems. We also need to clearly demonstrate how *in vivo* computers will outperform alternative technologies, and I suspect that here the complexity is the key. Increasing the size of synthetic networks to 10 and 20 elements will represent somewhat of a milestone, and this feat will necessarily be accompanied by explicit consideration of and dealing with random fluctuations. It may turn out that, not unlike in natural networks, the proportion of the true “computational” components will be quite low and the bulk of the circuit will be dedicated to maintaining robust, stable operation. In parallel, numerous technical issues will have to be solved, for example low-cost, rapid assembly of 20+ component circuits. Currently-available gene synthesis technologies are still costly and time-consuming. Another question pertains to the development of “debugging” tools, in other words, how will we know if such a complex network operates properly under all possible conditions? Yet another challenge is implanting these circuits in live cells, be it bacteria, yeast or mammalian cells. Each organism is different and each will attempt to bypass the newly introduced network. Having said that, there may be no alternative but to invest in overcoming these challenges, because in the long run this may be the only multi-purpose technology to reprogram cells, a task that is in high demand in all areas of biotechnology, bioengineering and biomedicine.

References

1. Bennett CH. International Journal of Theoretical Physics 1982;21:905–940.
2. Adleman LM. Science 1994;266:1021–1024. [PubMed: 7973651]
3. Lipton RJ. Science 1995;268:542–545. [PubMed: 7725098]
4. Ouyang Q, Kaplan PD, Liu SM, Libchaber A. Science 1997;278:446–449. [PubMed: 9334300]
5. Kari L. Mathematical Intelligencer 1997;19:9–22.
6. Frutos AG, Liu QH, Thiel AJ, Sanner AMW, Condon AE, Smith LM, Corn RM. Nucleic Acids Research 1997;25:4748–4757. [PubMed: 9441280]

7. Reif JH. Algorithmica 1999;25:142–175.
8. Jonoska N, Karl SA, Saito M. Biosystems 1999;52:143–153. [PubMed: 10636039]
9. Faulhammer D, Cukras AR, Lipton RJ, Landweber LF. Proceedings of the National Academy of Sciences of the United States of America 2000;97:1385–1389. [PubMed: 10677471]
10. Paun G. Journal of Computer and System Sciences 2000;61:108–143.
11. Liu QH, Wang LM, Frutos AG, Condon AE, Corn RM, Smith LM. Nature 2000;403:175–179. [PubMed: 10646598]
12. Head T, Rozenberg G, Bladergroen RS, Breek CKD, Lommerse PHM, Spaink HP. Biosystems 2000;57:87–93. [PubMed: 11004388]
13. USA Pat 6,266,569. 2001.
14. Turing AM. Proceedings of the London Mathematical Society 1937;42:230–265.
15. McCulloch WS, Pitts W. Bulletin of Mathematical Biology 1943;5:115–133.
16. Jacob F, Monod J. Journal of Molecular Biology 1961;3:318–&. [PubMed: 13718526]
17. Monod J, Jacob F. Cold Spring Harbor Symposia on Quantitative Biology 1961;26:389–&.
18. Monod J, Changeux JP, Jacob F. Journal of Molecular Biology 1963;6:306–&. [PubMed: 13936070]
19. Setty Y, Mayo AE, Surette MG, Alon U. Proceedings of the National Academy of Sciences of the United States of America 2003;100:7702–7707. [PubMed: 12805558]
20. Regev A, Shapiro E. Nature 2002;419:343–343. [PubMed: 12353013]
21. Dekel E, Alon U. Nature 2005;436:588–592. [PubMed: 16049495]
22. Mayo AE, Setty Y, Shavit S, Zaslaver A, Alon U. Plos Biology 2006;4:555–561.
23. Berg HC, Purcell EM. Biophysical Journal 1977;20:193–219. [PubMed: 911982]
24. Winfree E, Liu FR, Wenzler LA, Seeman NC. Nature 1998;394:539–544. [PubMed: 9707114]
25. Fu TJ, Seeman NC. Biochemistry 1993;32:3211–3220. [PubMed: 8461289]
26. Chen HL, Schulman R, Goel A, Winfree E. Nano Letters 2007;7:2913–2919. [PubMed: 17718529]
27. Fujibayashi K, Hariadi R, Park SH, Winfree E, Murata S. Nano Letters 2008;8:1791–1797. [PubMed: 18162000]
28. Rothemund PWK. Nature 2006;440:297–302. [PubMed: 16541064]
29. Mao CD, LaBean TH, Reif JH, Seeman NC. Nature 2000;407:493–496. [PubMed: 11028996]
30. Smith, WD. DNA-based computers Proceedings of a DIMACS workshop. Lipton, RJ.; Baum, EB., editors. American Mathematical Society; 1995. p. 121-186.Editon edn
31. Rothemund, PWK. DNA-based computers Proceedings of a DIMACS workshop. Lipton, RJ.; Baum, EB., editors. 1995. p. 75-120.Editon edn
32. Shapiro E, Benenson Y. Scientific American 2006;295:44–51. [PubMed: 16708487]
33. Sakamoto K, Gouzu H, Komiya K, Kiga D, Yokoyama S, Yokomori T, Hagiya M. Science 2000;288:1223–1226. [PubMed: 10817993]
34. Komiya K, Sakamoto K, Kameda A, Yamamoto M, Ohuchi A, Kiga D, Yokoyama S, Hagiya M. Biosystems 2006;83:18–25. [PubMed: 16343736]
35. Benenson Y, Paz-Elizur T, Adar R, Keinan E, Livneh Z, Shapiro E. Nature 2001;414:430–434. [PubMed: 11719800]
36. Benenson Y, Adar R, Paz-Elizur T, Livneh Z, Shapiro E. Proceedings of the National Academy of Sciences of the United States of America 2003;100:2191–2196. [PubMed: 12601148]
37. Adar R, Benenson Y, Linshiz G, Rosner A, Tishby N, Shapiro E. Proceedings of the National Academy of Sciences of the United States of America 2004;101:9960–9965. [PubMed: 15215499]
38. Benenson Y, Gil B, Ben-Dor U, Adar R, Shapiro E. Nature 2004;429:423–429. [PubMed: 15116117]
39. Soloveichik D, Winfree E. Theoretical Computer Science 2005;344:279–297.
40. Desilva AP, Gunaratne HQN, McCoy CP. Nature 1993;364:42–44.
41. Macdonald J, Stelanovic D, Stojanovic MN. Scientific American 2008;299:84–91. [PubMed: 18998351]
42. Ellington AD, Szostak JW. Nature 1990;346:818–822. [PubMed: 1697402]
43. Ellington AD, Szostak JW. Nature 1992;355:850–852. [PubMed: 1538766]
44. Robertson MP, Ellington AD. Nature Biotechnology 1999;17:62–66.

45. Tang J, Breaker RR. Chemistry & Biology 1997;4:453–459. [PubMed: 9224568]
46. Soukup GA, Breaker RR. Current Opinion in Structural Biology 2000;10:318–325. [PubMed: 10851196]
47. Stojanovic MN, de Prada P, Landry DW. Chembiochem 2001;2:411–415. [PubMed: 11828471]
48. Stojanovic MN, Mitchell TE, Stefanovic D. Journal of the American Chemical Society 2002;124:3555–3561. [PubMed: 11929243]
49. Stojanovic MN, Stefanovic D. Nature Biotechnology 2003;21:1069–1074.
50. Macdonald J, Li Y, Sutovic M, Lederman H, Pendri K, Lu WH, Andrews BL, Stefanovic D, Stojanovic MN. Nano Letters 2006;6:2598–2603. [PubMed: 17090098]
51. Stojanovic MN, Semova S, Kolpashchikov D, Macdonald J, Morgan C, Stefanovic D. Journal of the American Chemical Society 2005;127:6914–6915. [PubMed: 15884910]
52. Levy M, Ellington AD. Proceedings of the National Academy of Sciences of the United States of America 2003;100:6416–6421. [PubMed: 12743371]
53. Seelig G, Soloveichik D, Zhang DY, Winfree E. Science 2006;314:1585–1588. [PubMed: 17158324]
54. Mao CD, Sun WQ, Shen ZY, Seeman NC. Nature 1999;397:144–146. [PubMed: 9923675]
55. Yurke B, Turberfield AJ, Mills AP, Simmel FC, Neumann JL. Nature 2000;406:605–608. [PubMed: 10949296]
56. Yin P, Choi HMT, Calvert CR, Pierce NA. Nature 2008;451:318–U314. [PubMed: 18202654]
57. Zhang DY, Turberfield AJ, Yurke B, Winfree E. Science 2007;318:1121–1125. [PubMed: 18006742]
58. Sivan S, Lotan N. Biotechnology Progress 1999;15:964–970. [PubMed: 10585179]
59. Sivan S, Tuchman S, Lotan N. Biosystems 2003;70:21–33. [PubMed: 12753934]
60. Baron R, Lioubashevski O, Katz E, Niazov T, Willner I. Journal of Physical Chemistry A 2006;110:8548–8553.
61. Baron R, Lioubashevski O, Katz E, Niazov T, Willner I. Angewandte Chemie-International Edition 2006;45:1572–1576.
62. Niazov T, Baron R, Katz E, Lioubashevski O, Willner I. Proceedings of the National Academy of Sciences of the United States of America 2006;103:17160–17163. [PubMed: 17088533]
63. Privman V, Strack G, Solenov D, Pita M, Katz E. Journal of Physical Chemistry B 2008;112:11777–11784.
64. Bar-Ziv R, Tlusty T, Libchaber A. Proceedings of the National Academy of Sciences of the United States of America 2002;99:11589–11592. [PubMed: 12186973]
65. Ashkenasy G, Ghadiri MR. Journal of the American Chemical Society 2004;126:11140–11141. [PubMed: 15355081]
66. Noireaux V, Bar-Ziv R, Libchaber A. Proceedings of the National Academy of Sciences of the United States of America 2003;100:12672–12677. [PubMed: 14559971]
67. Sugita M. Journal of Theoretical Biology 1961;1:415–&. [PubMed: 13918223]
68. Sugita M. Journal of Theoretical Biology 1963;4:179–&. [PubMed: 5875160]
69. Sugita M, Fukuda N. Journal of Theoretical Biology 1963;5:412–&. [PubMed: 5336375]
70. Sugita M. Journal of Theoretical Biology 1966;13:330–&.
71. Sugita M. Journal of Theoretical Biology 1975;53:223–237. [PubMed: 1105009]
72. Hjelmfelt A, Weinberger ED, Ross J. Proceedings of the National Academy of Sciences of the United States of America 1991;88:10983–10987. [PubMed: 1763012]
73. Hjelmfelt A, Weinberger ED, Ross J. Proceedings of the National Academy of Sciences of the United States of America 1992;89:383–387. [PubMed: 11607249]
74. Arkin A, Ross J. Biophysical Journal 1994;67:560–578. [PubMed: 7948674]
75. Hjelmfelt A, Ross J. Physica D 1995;84:180–193.
76. Beer MA, Tavazoie S. Cell 2004;117:185–198. [PubMed: 15084257]
77. Segal E, Raveh-Sadka T, Schroeder M, Unnerstall U, Gaul U. Nature 2008;451:535–U531. [PubMed: 18172436]
78. Weiss, R.; Homsy, GE.; Knight, TF. Evolution as Computation: DIMACS Workshop. Landweber, LF.; Winfree, E., editors. Springer; 1999. p. 275-295.Editon edn

79. Buchler NE, Gerland U, Hwa T. Proceedings of the National Academy of Sciences of the United States of America 2003;100:5136–5141. [PubMed: 12702751]
80. Gardner TS, Cantor CR, Collins JJ. Nature 2000;403:339–342. [PubMed: 10659857]
81. Elowitz MB, Leibler S. Nature 2000;403:335–338. [PubMed: 10659856]
82. Becskei A, Serrano L. Nature 2000;405:590–593. [PubMed: 10850721]
83. Basu S, Gerchman Y, Collins CH, Arnold FH, Weiss R. Nature 2005;434:1130–1134. [PubMed: 15858574]
84. Ajo-Franklin CM, Drubin DA, Eskin JA, Gee EPS, Landgraf D, Phillips I, Silver PA. Genes & Development 2007;21:2271–2276. [PubMed: 17875664]
85. Canton B, Labno A, Endy D. Nature Biotechnology 2008;26:787–793.
86. Tigges M, Marquez-Lago TT, Stelling J, Fussenegger M. Nature 2009;457:309–312. [PubMed: 19148099]
87. Levskaya A, Chevalier AA, Tabor JJ, Simpson ZB, Lavery LA, Levy M, Davidson EA, Scouras A, Ellington AD, Marcotte EM, Voigt CA. Nature 2005;438:441–442. [PubMed: 16306980]
88. Anderson JC, Clarke EJ, Arkin AP, Voigt CA. Journal of Molecular Biology 2006;355:619–627. [PubMed: 16330045]
89. Kramer BP, Fischer C, Fussenegger M. Biotechnology and Bioengineering 2004;87:478–484. [PubMed: 15286985]
90. Bronson JE, Mazur WW, Cornish VW. Molecular Biosystems 2008;4:56–58. [PubMed: 18075675]
91. Sayut DJ, Niu Y, Sun LH. Applied and Environmental Microbiology 2009;75:637–642. [PubMed: 19060164]
92. Hooshangi S, Thiberge S, Weiss R. Proceedings of the National Academy of Sciences of the United States of America 2005;102:3581–3586. [PubMed: 15738412]
93. Blake WJ, Kaern M, Cantor CR, Collins JJ. Nature 2003;422:633–637. [PubMed: 12687005]
94. Shen-Orr SS, Milo R, Mangan S, Alon U. Nature Genetics 2002;31:64–68. [PubMed: 11967538]
95. Dueber JE, Yeh BJ, Chak K, Lim WA. Science 2003;301:1904–1908. [PubMed: 14512628]
96. Bashor CJ, Helman NC, Yan SD, Lim WA. Science 2008;319:1539–1543. [PubMed: 18339942]
97. Skerker JM, Perchuk BS, Siryaporn A, Lubin EA, Ashenberg O, Goulian M, Laub MT. Cell 2008;133:1043–1054. [PubMed: 18555780]
98. Rackham O, Chin JW. Journal of the American Chemical Society 2005;127:17584–17585. [PubMed: 16351070]
99. Chin JW. Nature Chemical Biology 2006;2:304–311.
100. Isaacs FJ, Dwyer DJ, Collins JJ. Nature Biotechnology 2006;24:545–554.
101. Davidson EA, Ellington AD. Nature Chemical Biology 2007;3:23–28.
102. Mandal M, Boese B, Barrick JE, Winkler WC, Breaker RR. Cell 2003;113:577–586. [PubMed: 12787499]
103. Winkler WC, Breaker RR. Annual Review of Microbiology 2005;59:487–517.
104. Isaacs FJ, Dwyer DJ, Ding CM, Pervouchine DD, Cantor CR, Collins JJ. Nature Biotechnology 2004;22:841–847.
105. Sudarsan N, Hammond MC, Block KF, Welz R, Barrick JE, Roth A, Breaker RR. Science 2006;314:300–304. [PubMed: 17038623]
106. Fire A, Xu SQ, Montgomery MK, Kostas SA, Driver SE, Mello CC. Nature 1998;391:806–811. [PubMed: 9486653]
107. Lee Y, Kim M, Han JJ, Yeom KH, Lee S, Baek SH, Kim VN. Embo Journal 2004;23:4051–4060. [PubMed: 15372072]
108. Stegmeier F, Hu G, Rickles RJ, Hannon GJ, Elledge SJ. Proceedings of the National Academy of Sciences of the United States of America 2005;102:13212–13217. [PubMed: 16141338]
109. Bayer TS, Smolke CD. Nature Biotechnology 2005;23:337–343.
110. Win MN, Smolke CD. Proceedings of the National Academy of Sciences of the United States of America 2007;104:14283–14288. [PubMed: 17709748]
111. Win MN, Smolke CD. Science 2008;322:456–460. [PubMed: 18927397]

112. Rinaudo K, Bleris L, Maddamsetti R, Subramanian S, Weiss R, Benenson Y. Nature Biotechnology 2007;25:795–801.
113. An CI, Trinh VB, Yokobayashi Y. Rna-a Publication of the Rna Society 2006;12:710–716.
114. Tuleuova N, An CI, Ramanculov E, Revzin A, Yokobayashi Y. Biochemical and Biophysical Research Communications 2008;376:169–173. [PubMed: 18765226]
115. Beisel CL, Bayer TS, Hoff KG, Smolke CD. Molecular Systems Biology 2008;4
116. Anderson JC, Voigt CA, Arkin AP. Molecular Systems Biology 2007;3:8.

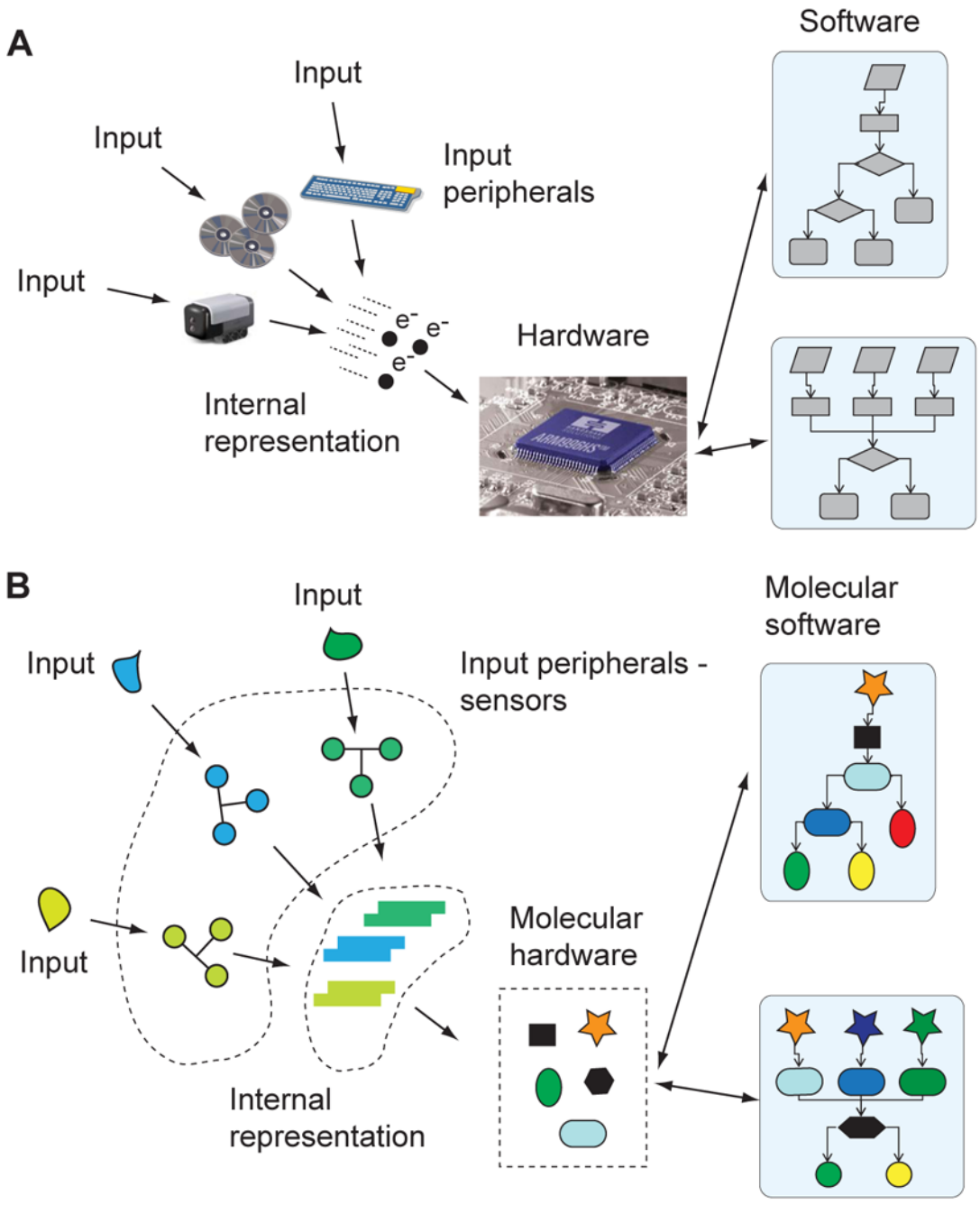

Figure 1.
Silicon-based and biomolecular computers compared. **A**, General layout of a conventional computer. **B**, General layout of a biomolecular computer. Molecular inputs are converted into an internal molecular representation via a number of molecular sensors. Those molecules are in turn computationally processed by “molecular software” and “hardware”. Molecular hardware would normally contain some invariant mechanisms (*e. g.* specific regulatory pathways) that can be easily reconfigured to implement different algorithms.

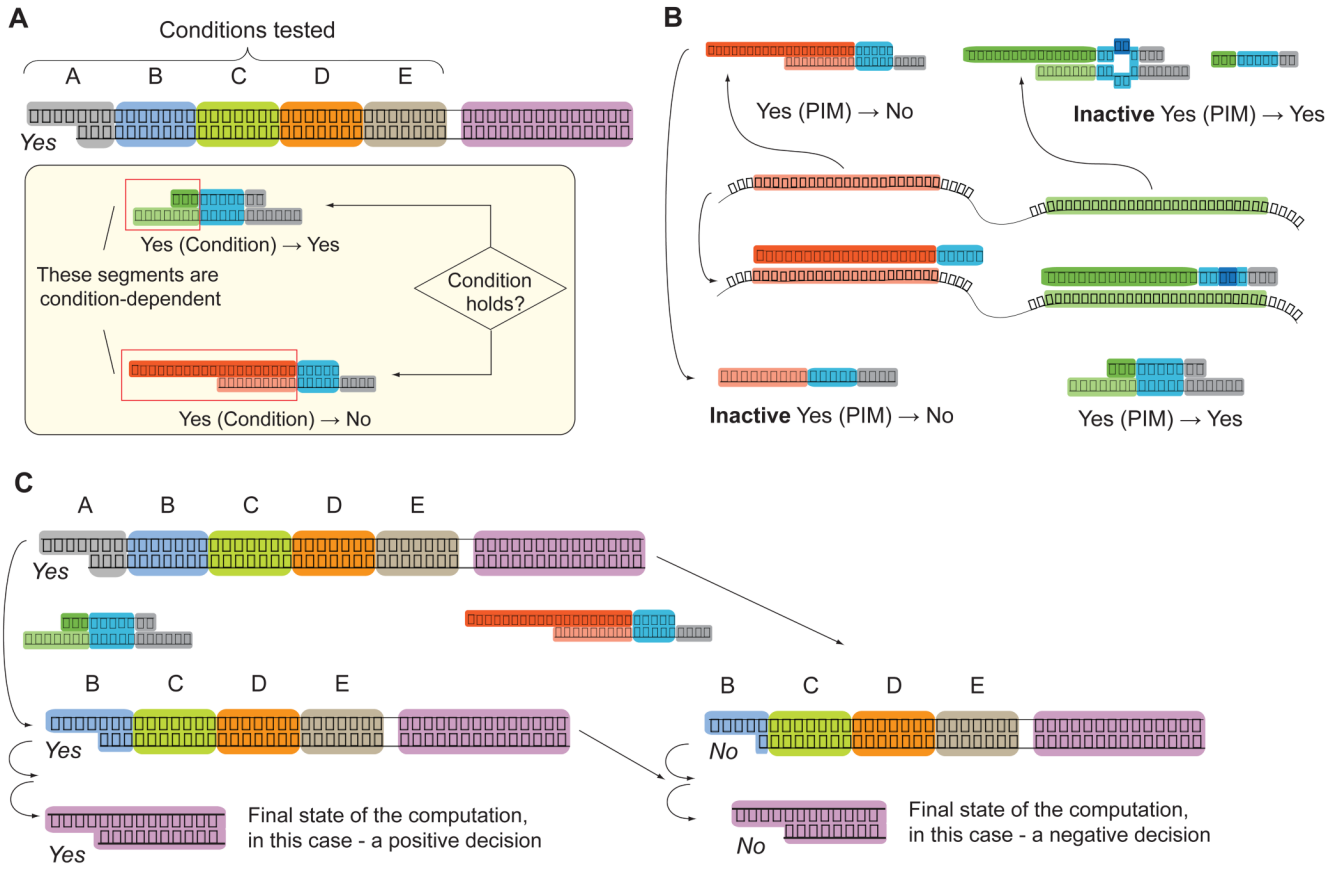

Figure 2.

A decision-making finite state machine. **A**, General layout of the system. A DNA molecule (top) encodes internal representation of the conditions to be tested, and the left-most segment is labeled with a single-stranded extrusion indicating the current state. A condition-checking "black box" that is external to the computation and constitutes input peripherals produces one of the two "transition molecules", depending on the condition state. **B**, The workings of the input peripherals. An mRNA molecule whose concentration is a condition of interest physically interacts with appropriately designed transition molecules and alters their composition in concentration-dependent manner using strand exchange mechanism. **C**, The underlying chemical process: transition molecules generated by the input peripherals chew on the DNA molecule and move it to the next step, generating either a Yes-state or No-state encoding sticky end. The newly exposed sticky ends are substrates for the next round of "chop-off" with a pair of applicable transition molecules, and so on until the final state is reached. The sequence exposed in this final configuration triggers more downstream processing resulting in a release of drug-like molecule (not shown).

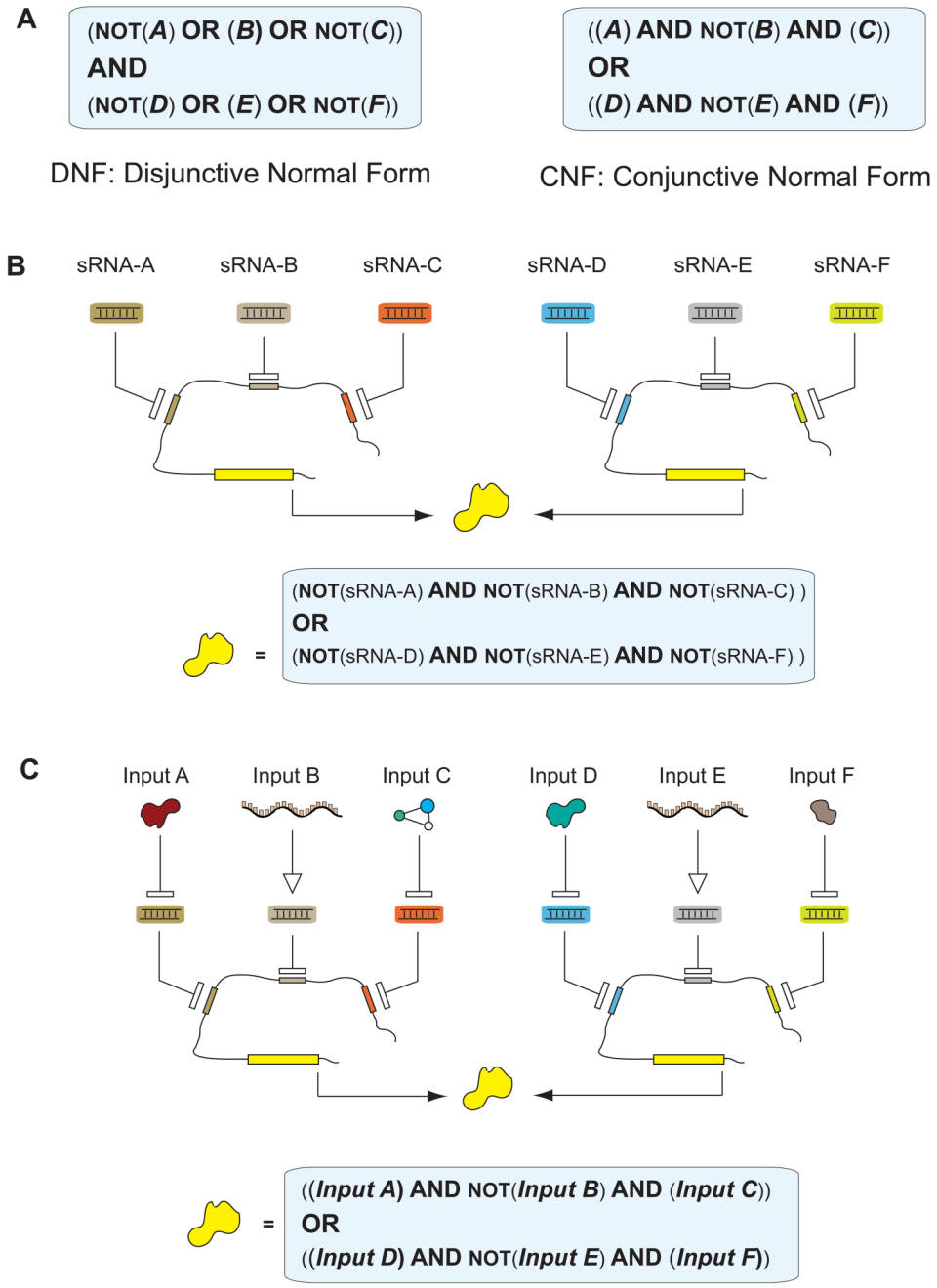

Figure 3.

Biocomputers that employ normal logic forms. **A**, Disjunctive (DNF) and conjunctive (CNF) logic forms **B**, The internal structure of an RNAi-based biocomputer and the core logic function it computes. The function can be reprogrammed by adding or removing small RNA targets as well as moving the targets among the mRNA reporter constructs. The same target can appear in multiple reporter constructs as well. **C**, Computing logic relations between biological inputs once the input peripherals are established.