Available online at www.sciencedirect.com
ScienceDirect
Physica D 237 (2008) 1165–1172
PHYSICA D
www.elsevier.com/locate/physd

Towards molecular computers that operate in a biological environment

Maya Kahan[a], Binyamin Gil[a], Rivka Adar[a], Ehud Shapiro[a,b,*]

[a] *Department of Biological Chemistry, Weizmann Institute of Science, Rehovot 76100, Israel*
[b] *Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel*

Available online 8 February 2008

Abstract

Even though electronic computers are the only computer species we are accustomed to, the mathematical notion of a programmable computer has nothing to do with electronics. In fact, Alan Turing's notional computer [L.M. Turing, On computable numbers, with an application to the entcheidungsproblem, Proc. Lond. Math. Soc. 42 (1936) 230–265], which marked in 1936 the birth of modern computer science and still stands at its heart, has greater similarity to natural biomolecular machines such as the ribosome and polymerases than to electronic computers. This similarity led to the investigation of DNA-based computers [C.H. Bennett, The thermodynamics of computation — Review, Int. J. Theoret. Phys. 21 (1982) 905–940; A.M. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (1994) 1021–1024]. Although parallelism, sequence specific hybridization and storage capacity, inherent to DNA and RNA molecules, can be exploited in molecular computers to solve complex mathematical problems [Q. Ouyang, et al., DNA solution of the maximal clique problem, Science 278 (1997) 446–449; R.J. Lipton, DNA solution of hard computational problems, Science 268 (1995) 542–545; R.S. Braich, et al., Solution of a 20-variable 3-SAT problem on a DNA computer, Science 296 (2002) 499–502; Liu Q., et al., DNA computing on surfaces, Nature 403 (2000) 175–179; D. Faulhammer, et al., Molecular computation: RNA solutions to chess problems, Proc. Natl. Acad. Sci. USA 97 (2000) 1385–1389; C. Mao, et al., Logical computation using algorithmic self-assembly of DNA triple-crossover molecules, Nature 407 (2000) 493–496; A.J. Ruben, et al., The past, present and future of molecular computing, Nat. Rev. Mol. Cell. Biol. 1 (2000) 69–72], we believe that the more significant potential of molecular computers lies in their ability to interact directly with a biochemical environment such as the bloodstream and living cells. From this perspective, even simple molecular computations may have important consequences when performed in a proper context. We envision that molecular computers that operate in a biological environment can be the basis of "smart drugs", which are potent drugs that activate only if certain environmental conditions hold. These conditions could include abnormalities in the molecular composition of the biological environment that are indicative of a particular disease. Here we review the research direction that set this vision and attempts to realize it.

Keywords: Turing machine; Finite automaton; Molecular computer; DNA computing; Smart drug

1. From Turing machines to molecular computers

In 1936 Alan Turing conceived of the Turing machine [1], a notional rule-based device that moves over a potentially limitless tape with symbols written on it and can read, write and rewrite these symbols. The Turing machine marks the beginning of modern computer science and still stands at its heart, as a provably universal model of computation. A decade later, John von Neumann described the architecture of the first practical programmable computer [11]. It made use of electrical implementation of Boolean logic circuits by realizing "0" and "1" as the absence or presence of electrical signals. It was only decades later that scientists began to realize [2] that natural biomolecular processes within living cells, such as DNA duplication, transcription and translation, realize Turing machine-like information processing operations using DNA, RNA and enzymes. Similarly to the Turing machine, in these processes an input string is processed in a stepwise manner adding symbols according to fixed rules. This knowledge encouraged researchers in the field of biomolecular computing [2,3] to use biomolecules (DNA, RNA and enzymes) to construct programmable molecular computers. DNA is composed of four building blocks A, C, T and G termed nucleotides or bases. They are covalently strung

* Corresponding author at: Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel.
E-mail address: ehud.shapiro@weizmann.ac.il (E. Shapiro).

0167-2789/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.physd.2008.01.027

together to form a directional strand, which can specifically bind a complementary strand (C to G and A to T) in an anti-parallel manner, named hybridization, forming double-stranded DNA (dsDNA; one strand is called forward or sense strand and the other is called reverse or anti-sense strand). In nature, DNA is located in the cell's nucleus; it is double-stranded DNA and carries the genetic information applied during development and function of almost all known living organisms.

Molecular computers [3–10,12–17] typically use synthetic DNA that is chemically synthesized as a single-stranded DNA (ssDNA). To manipulate DNA molecules, many molecular computers operate by using a diversity of enzymes. Some enzymes, called nucleases, digest dsDNA molecules by cleaving the covalent bonds between two adjacent nucleotides while other enzymes (ligases) can ligate two dsDNA molecules by forming covalent bonds between them. Some nucleases cleave DNA only in specific locations in a sequence-dependant manner, while others can cleave any DNA molecule at any location. In addition, cleavage of dsDNA can split it into two double-stranded DNA molecules each with a short sequence overhang named 'sticky-ends'. Those 'sticky-ends' can bind, or stick to complementary sequences. Other enzymes will cleave the dsDNA in a blunt manner, meaning no single-stranded overhangs will be formed upon digest of one dsDNA into two separate dsDNA molecules.

The field of DNA computing began by attempts to exploit the parallelism, the sequence specificity of hybridization and the storage capacity of DNA molecules to solve complex computational problems.

Another area related to DNA computing is 'Synthetic biology', which utilizes artificial genetic circuits that have a potential to become important tools for controlling cellular behavior and studying biomolecular systems [20]. Relatively simple artificial networks, including feedback systems [21–23], toggle switches [24,25], oscillators [25–28] and cell–cell communication systems [29], were constructed using predictive models that uncovered network behavior and helped guiding experimental design [30]. Artificial genetic circuits could act, at the genetic level, as tiny "programs" though control or monitor in specific manner cellular behavior, providing various potential applications in biotechnology, medicine, environmental science and other areas [20,31].

2. Molecular computers solve computational problems

In 1994, Adleman and co-workers realized the first concrete molecular computer [3] that can solve the Hamiltonian path problem that is related to the famous traveling salesman problem. It is a member of the family of the so-called NP-Complete problems, which have so far defied polynomial-time solutions on conventional computers. He discovered a way to harness the power of DNA to solve this problem, finding the shortest path from start to end by going only once through all the points (cities). In Adleman's method each DNA molecule represents a directed edge (legal path between two points), and employs a chemical reaction that uses these DNA molecules as input to generate a combinatorial library of DNA molecules that represent all possible legal paths from any two points. From this combinatorial library a DNA molecule representing the correct solution was obtained by a series of biochemical steps employing standard molecular biology techniques. Since then other NP-complete problems have been solved by similar methods [4–10]. In 2003, Stojanovic et al., have successfully implemented a DNA-based computer that can play tic-tac-toe against a human player and never lose [12].

Some researchers in the field believe that the parallelism, low energy consumption and information density that characterize molecular computers could be used to attack computational problems like NP-complete problems, which resisted conventional methods. However, difficulties in scaling up DNA-based solutions for computational problems gave rise to the opposite opinion that DNA computing would never be able to compete directly with silicon-based technology. Today the general notion is that the true potential of molecular computers lies in their ability to directly interact with the biochemical environment. This ability suggests the vision of an autonomous molecular computer that can interact with endogenous biological molecules, check for disease indicators, perform diagnosis based on these indicators according to programmed medical knowledge and administer *in vivo*, upon positive diagnosis, the requisite drug [17–19].

3. Molecular computers that interact with a biological environment

Interaction between a computer and a biological environment may be possible if the computer uses components similar to those naturally existing in the cell, e.g. DNA, RNA, and enzymes. Molecular computers use these molecules as their software, hardware, input and output. Molecular computers can operate *in vitro*, in a laboratory vessel (tube) or other controlled experimental environment or *in vivo*, occurring within the complex environment of a living organism or natural setting. We review several examples of such computing devices.

4. A programmable autonomous finite automaton that solves simple computational problems *in vitro*

A finite automaton is a simplified Turing machine that can only read, not write, on its input and can move only in one direction. The input is a sequence of symbols in which their interpretation depends on the application. The machine can be in one of a finite number of internal states; of which one is designated an initial state and some are designated accepting states. Its software consists of transition rules, each specifying a next state based on the current state and the current symbol. It is initially positioned on the leftmost input symbol in the initial state. In each transition the machine moves one symbol to the right, changing its internal state according to one of the applicable transition rules. Alternatively, it may 'suspend' without completing the computation if no transition rule applies. A computation terminates on processing the last input symbol. An automaton is said to accept an input if a computation on this input terminates

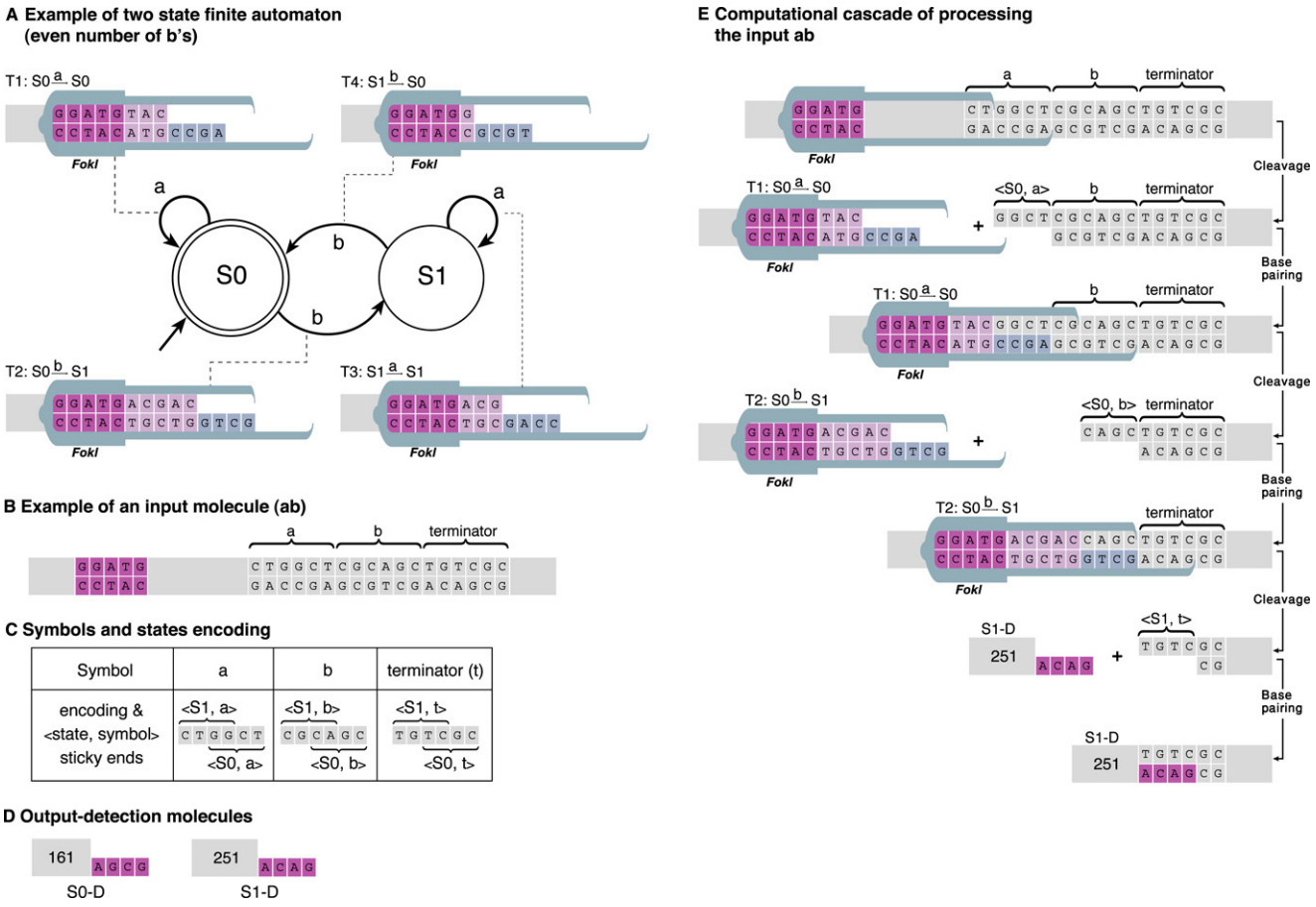

Fig. 1. An example of a two state finite automaton accepting inputs with even number of b's. (A) Diagram representing an automaton with two states, S0 and S1, and input alphabet a and b. Incoming straight arrow represents the initial state. Labeled arrows represent transition rules. The double circle represents the accepting state (S0). The sticky-end of a transition molecule detects the current state and symbol by hybridizing with the sticky-end of the input, and determines the next state by directing the attached enzyme FokI to cleave the input molecule in a specific position with the next symbol. The transition molecule consists of a sticky-end that detects a ⟨state, symbol⟩ combination (blue), a *Fok*I recognition site (dark red) and spacer (pink) that determines the location of *Fok*I's cleavage position inside the next symbol. This position, in turn, will define the next state. Transitions with one base pair (bp) spacers transfer from S1 to S0, 3-bp maintain the current state, and 5-bp transfer S0 to S1. (B) Example of input molecule that encodes the string ab. (C) The encoding for the input symbols a, b, and terminator (sense strands) and the sequences of the ⟨state, symbol⟩ sticky-ends. (D) Structure of the output-detection molecules. (E) Computation cascade of processing the input molecule ab.

in an accepting final state [13]. In 1995, Rothemund described, without implementing, a DNA-based Turing machine [14]. In 2003 Benenson et al. [15], have designed and implemented a two states, two symbols finite automaton that uses dsDNA molecules as input and transitions and a DNA-manipulating enzyme, the bacterial nuclease FokI as hardware [32]. An example of a two state finite automaton that accepts only input strings with a's and b's that have an even number of b's is shown in Fig. 1.

The automaton's input string is realized by a dsDNA molecule that encodes the input symbols and a terminator that signals the end of the string. Fig. 1(B) represents an example of input molecule that encodes the string ab. Each symbol is realized by a unique sequence of 6 base pairs (bp) consisting of two overlapping 4-bp frames: the leftmost frame encodes the symbol combined with the state S0 and rightmost frame encodes the symbol combined with the state S1 (Fig. 1(C)). Upon cleavage, the sense strand of one of the frames will be exposed; forming a 4-nucleotides sticky-end represents the state and symbol. Transition rules are realized by a short dsDNA molecule. Each transition molecule is composed of four regions: (1) inert dsDNA tail; (2) FokI recognition/binding site; (3) spacer region of zero to five base pairs; (4) four-nucleotides sticky-end, comprised of the anti-sense strand. The system also contains output-detecting molecules of different lengths (Fig. 1(D)), each of which can interact selectively with a different output molecule to form the output-reporting molecule that indicates the final state and can be readily detected by gel electrophoresis.

The computation is initiated by input cleavage by FokI, revealing four nucleotides sticky-end on the sense strand. This sticky-end represents the initial state of the automaton (S0) and the first inputs symbol. The computation proceeds via a cascade of transition cycles. In each cycle, a transition molecule that possesses a complementary sticky-end to the input molecule will hybridize to it, followed by FokI cleavage inside the next symbol resulting in exposure of a new four-nucleotide sticky-end. The length of the transition spacer determines the cleavage site of FokI inside the next input symbol, hence exposing one of the two frames' sense strand encoding the next state

and symbol. The computation proceeds until no transition molecule matches the exposed sticky-end of the input or until the terminator symbol is cleaved, forming an output molecule that encodes the final state. In Fig. 1(E), for example, the ab input is not accepted, since the final state is S1 which is a non-accepting state.

A micro-liter of computation mixture holds close to three trillion automata that can operate in parallel and independently. A computation over a 4-symbol-long input, at room temperature, rendered output-reporting molecules with 50% yield, in approximately 1 h. As for energy consumption, in each transition two ATP molecules were consumed, releasing 1.5×10^{-19} J. Multiplying this number by the transition rate (10^9 transitions per second) provides an energy consumption rate of 10^{-10} W, with accuracy of 99.8% per transition.

The automaton was shown to operate with several software programs on various inputs. In this design [15], unlike an earlier design [13], the transition molecules hybridize to the input molecule without ligation. The cleavage of the input molecule drives the computation forward by increasing entropy and releasing heat and, since software molecules are recycled, a fixed amount of software and hardware molecules can, in principle, process any input molecule of any length without external energy supply, except perhaps for garbage collection. Scaling up this automaton by the number of symbols and/or states is limited to the number of non-palindromic sticky-ends that can be designed and the length of the transition's spacer, respectively. Sticky-end limitations allow several tens of symbols. The characteristics of *Fok*I enable a finite machine with small number of states. Despite the performance advantages of this automaton, bacterial enzyme may not work well in cells of higher organisms, hindering its practical application.

The discovery of new enzymes able to digest dsDNA molecules resulting in longer sticky-ends and/or able to cut dsDNA far deep in the dsDNA molecule or engineering of the existing enzymes might allow the construction of automata with increased complexity, as well as enable better operation in eukaryotic cells. An example to a slightly more complex finite automaton was shown by Keinan and co-workers. They have shown a 3-symbol-3-state finite automaton using the BbvI and T4 DNA Ligase enzymes as hardware [33].

The fact that computations do not consume software molecules allowed Adar et al. to extend the range of applications of the molecular automaton from deterministic to stochastic computations [16]. A stochastic automaton ascribes each pair of competing transition rules (e.g. $S0 \xrightarrow{a} S0$ and $S0 \xrightarrow{a} S1$) two probabilities the sum of which is 1. The output of a stochastic computation is the probability to obtain each final state, computed by summing the probabilities of all possible computation paths that result in the same final state. Stochastic automata are useful for the analysis of sequences or processes that are not deterministic. A stochastic molecular automaton realizes the intended probability of each transition by the relative concentration of the software molecule encoding that transition. The results of Adar et al. show robustness of programmed transition probabilities to input molecule concentrations and to absolute software molecule concentrations and a good fit between predicted and actual probabilities of multi-step computations.

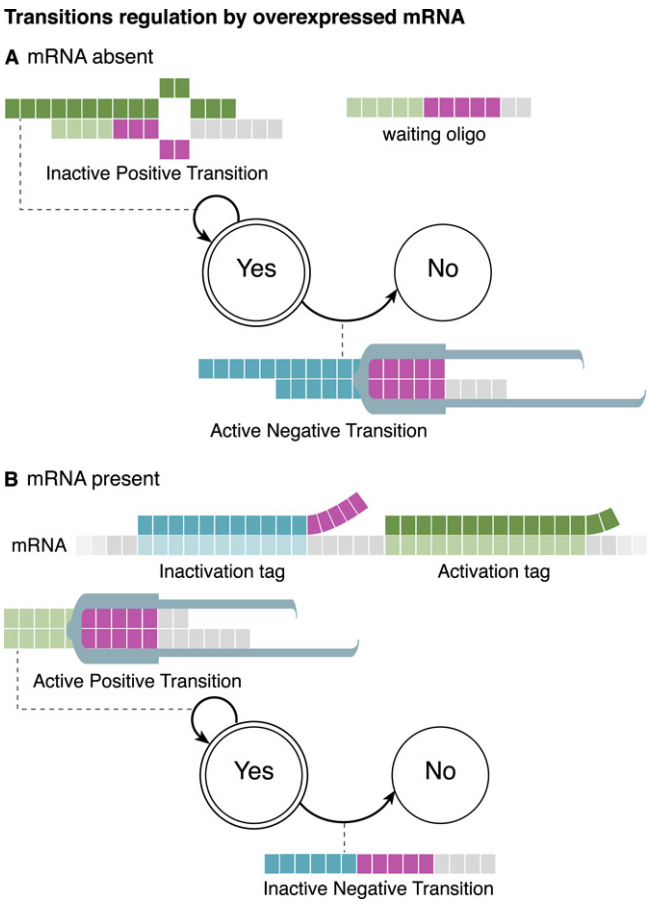

Fig. 2. Regulation of the automaton's transitions by an overexpressed mRNA species. (A) Absence of an overexpressed mRNA (disease indicator is absent) will result in the original transition conformation in which the positive transition molecule (Yes→Yes) is inactive and the negative transition molecule (Yes→No) is active. (B) When the mRNA is overexpressed (disease indicator is present), the 'inactivation tag' which is a short open region in the mRNA molecule (light blue) displaces the sense strand of the active negative transition molecule (Yes→No) thus destroying its activity. The 'activation tag' of the mRNA, which is another short open region on the same mRNA molecule (light green) displaces the sense strand of the inactive positive transition molecule (Yes→Yes), thus enabling the "waiting oligo", which is the correct transition sense strand, to hybridize to the anti-sense strand — resulting in an active transition. Strand that fades out represents longer, not shown, RNA.

5. Diagnostic computation of mRNA, *in vitro*

Benenson et al. used this stochastic automaton to logically analyze, *in vitro*, the levels of messenger RNA (mRNA) species [17]. mRNA is an RNA molecule that encodes a chemical "blueprint" for protein production. Expression levels of specific set of mRNAs can diagnose the presence or absence of a disease. Benenson et al. implemented successfully, *in vitro*, a diagnostic two state finite automaton that uses mRNA molecules as input, stochastically processes their levels, and upon positive diagnosis administers an active drug as output.

The automaton consists of three programmable modules: (1) input module, by which specific mRNA or mutated mRNA

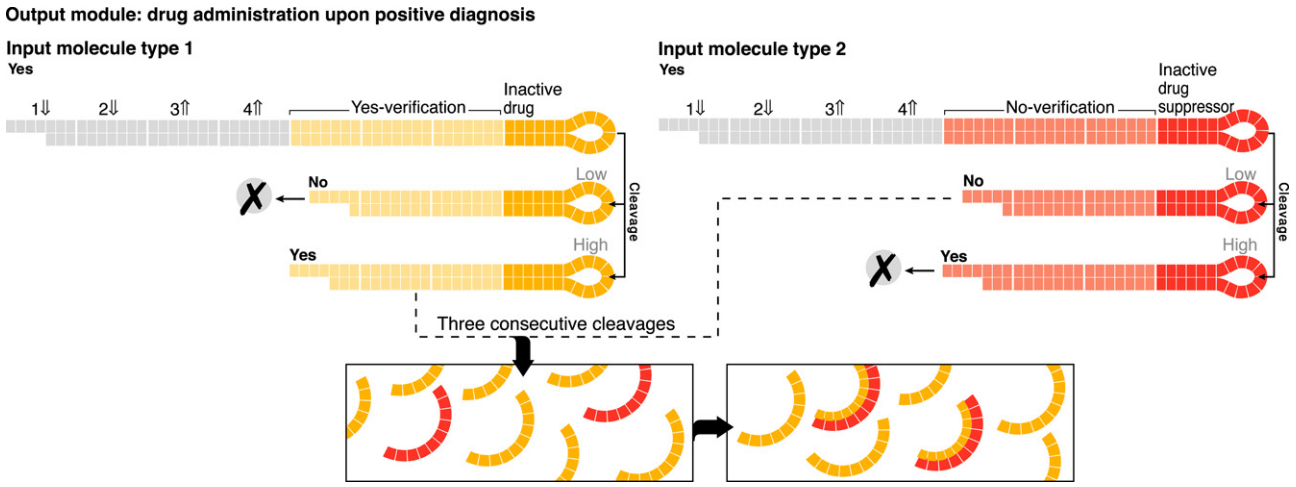

Fig. 3. Controlled release of the active drug, short single-stranded DNA molecule, upon positive diagnosis. Two types of input molecules that share the same diagnostic moiety (gray) but differ in their output moiety were exploited. Each of the "output moieties have a stem-loop structure. The stem (light colored) holds the functional part (dark colored) inactive. Upon positive diagnosis, the diagnostic moiety in both inputs will end with high concentration of Yes state and low concentration of No state. Transition-like molecules will cleave the stems of the input molecules that contain the drug, only if their diagnostic moiety cleavage ended in a Yes state. Different transition-like molecules will cleave the stems of the input molecules that contain the drug-suppressor, only if their diagnostic moiety cleavage ended in a No state. Drug-suppressor molecules would then suppress drug molecules in an equimolar manner. This will result in access of free drug molecules that will then be active. Changing the ratio between these two inputs will determine the diagnostic confidence level above which the drug will be released.

levels regulate the automaton's transition probabilities; (2) computational module, which is a stochastic one, and (3) an output module, capable of controlling the release of an active drug. As a proof of concept Benenson et al. programmed the computer to identify and analyze mRNAs of disease-related genes associated with small-cell lung cancer (SCLC) and prostate cancer (PC), and to produce a short ssDNA molecule functioning as anti-cancer drug [34]. The computer's operation is governed by a 'diagnostic rule' that encodes medical knowledge in simplified form. The left-hand side of the diagnostic rule encodes a conjunction of specific mRNA conditions (under-expression/over-expression/mutation). The right-hand side of the rule contains the drug to be released if all the conditions hold. The released drug is ssDNA, which inhibits the synthesis of an oncogenic protein by binding to its mRNA (a drug for the specific set of conditions tested) [34]. The computer's design allows any sufficiently long RNA molecule to function as a molecular indicator and any short ssDNA molecule, up to at least 21 nucleotides, to serve as the output drug.

The computation begins in the Yes state and checks one condition at a time. If a condition holds, the automaton remains in the Yes state; otherwise, the automaton changes its state to No and remains in that state for the rest of the computation. The input module adjusts the automaton's transitions probabilities by specific mRNA levels or by point mutated mRNAs. The probability of each transition is regulated by a specific mRNA expression condition, so that presence of an over-expressed mRNA increases the probability of a positive transition and decreases the probability of its competing negative transition, and vice versa if the mRNA level is normal (Fig. 2). Alternatively, normal expression of an under-expressed mRNA decreases the probability of a positive transition and increases the probability of its competing negative transition, and vice versa if the indicator is absent. This regulation is achieved by a displacement process; DNA strand detaches from its complementary strand to hybridize with the mRNA that offers a longer and energetically more favorable complementary region.

The stochastic behavior of the automaton is governed by the confidence in the presence of each indicator, so that the probability of a positive diagnosis is a result of the probabilities of the positive transitions for each of the indicators processed. By changing the ratio between positive and negative transitions of a particular indicator one can fine-tune the sensitivity of a diagnosis to the presence of its indicator. Instead of releasing a drug molecule on positive diagnosis and do nothing on negative diagnosis, Benenson et al. designed two types of input molecules (Fig. 3); one, will release drug molecule on a Yes result and do nothing on a No result and the other will release drug-suppressor molecule on a No result and do nothing on a Yes result. The drug-suppressor is ssDNA molecule with a sequence complementary to the drug molecule. Upon negative diagnosis, the drug-suppressor molecule will hybridize to the drug molecule, thus preventing its activity. The ratio between the released drug and drug-suppressor molecules determines the final drug concentration. This allows fine control over the diagnosis confidence threshold beyond which an active drug is administered.

The operation of this bio-molecular finite automaton *in vivo* has yet to be demonstrated.

6. Boolean logic using micro-RNAs as input, *in vitro*

Winfree and colleagues realized *in vitro* DNA-based logic gates and circuits to diagnose levels of micro-RNA (miRNA) molecules. miRNAs, are short, single-stranded, non-coding RNAs, 21-23 nucleotides long that negatively regulate gene expression. A logic gate performs logical operation on one or

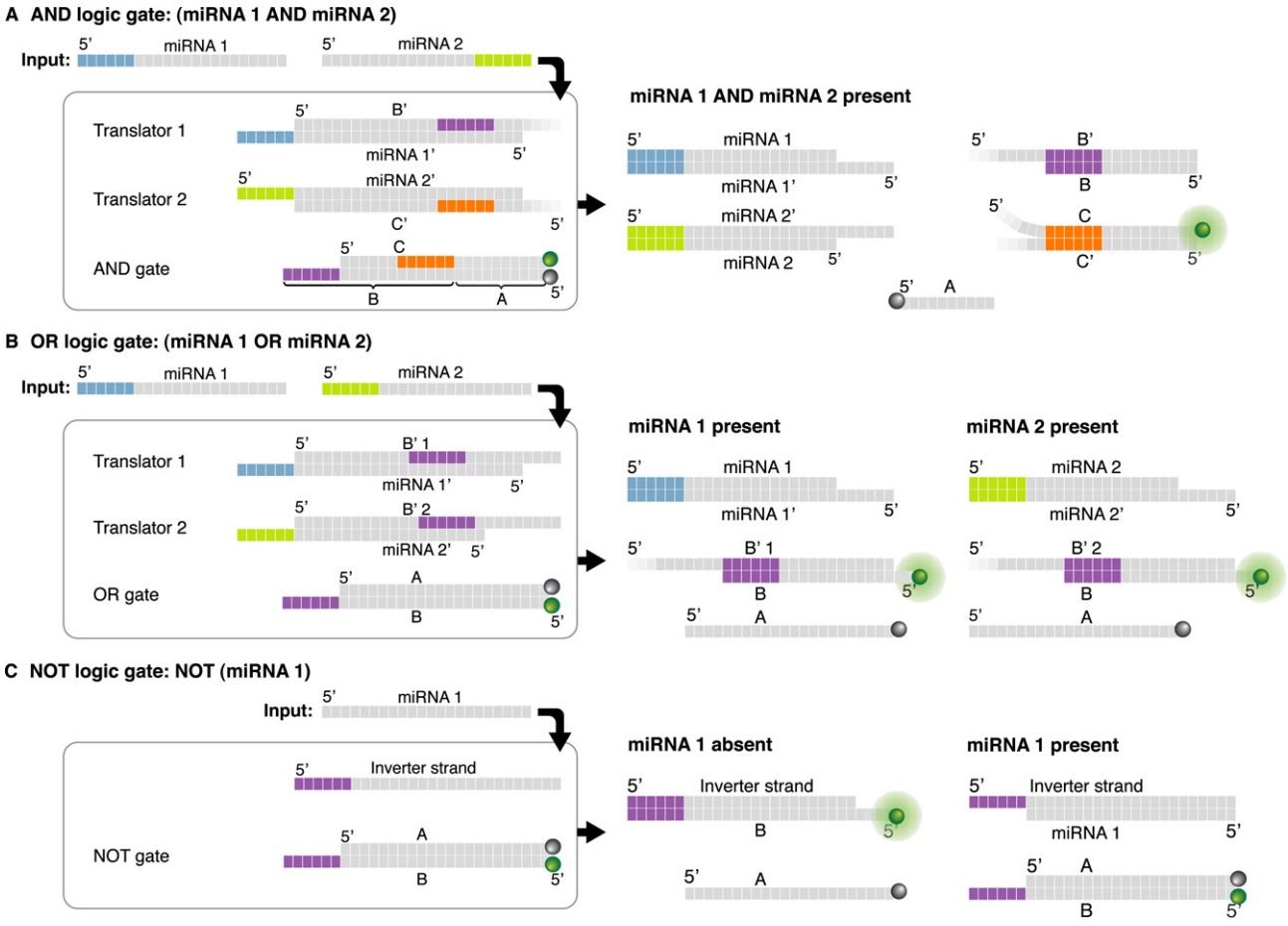

Fig. 4. Boolean logic gates are realized by DNA molecules. Regions that trigger displacement reactions are marked with the same color. (A) AND gate of two miRNA inputs. It consists of two translators and one AND gate. Each miRNA input will displace a translator's output strand. Only if both "translator output" strands will be released, they will displace the gate output strand, increasing the fluorescence intensity. (B) OR gate of two miRNA inputs. It consists of two translators and one OR gate. Each miRNA input can separately displace a distinct translator, but both translators would produce the same output strand. The final output gate (OR) could then be displaced by each of the translator outputs. (C) NOT gate was implemented by using a one input AND gate and an additional strand that triggers the gate (called inverter). The input strand acts as a competitive inhibitor, thus when present it will block the inverter strand and the gate would give a negative result, and vice versa. Strand that fades out represents longer, not shown, RNA.

more logic inputs and produces a single-logic output. Logic gates are the building blocks of digital circuits. Combinations of these logic gates generate circuits designed for a specific task. Winfree and colleagues implemented a complete set of Boolean logic functions: AND, OR and NOT (Fig. 4) [35].

The gates function without enzymes. Rather, their operation is based on strand displacement, where a free strand interferes with a double-stranded DNA molecule by pairing with one of its strands, causing its other strand to break loose. This simple design, although essential when conducting computations in biomolecular environment, is time consuming in the range of hours, therefore might exceed the biological relevant timescale. AND gate was implemented by dsDNA assembled by three complementary ssDNAs, an output strand and two gate strands (Fig. 4(A)). Each gate strand contains a recognition region that is complementary to its input. The strands were designed to assure output release only upon presence of the two gates inputs. OR gate was implemented by using two gates that produce the same output (Fig. 4(B)). NOT gate was implemented by using an additional strand that triggers the gate (called inverter), unless the input is present to act as a competitive inhibitor (Fig. 4(C)). Since NOT gate is implemented by additional strand that is added with the input, it is restricted to the first layer of the circuit. Multi-layer circuits are achieved by using ssDNA both as input and output. Output strand of each gate serves as an input to a downstream gate or as an evaluator. The strands are fluorescently labeled to provide a simple readout in a variable mode imager, e.g., Typhoon (Amersham). Signal restoration was attained by using amplifier gates; one input strand releases more than one output strand. Leaks were reduced by using thresholds, gate that requires presence of more than one copy of the input.

Winfree and colleagues realized successfully multi-layer circuits, constructed of five layer circuits consisting of 11 gates and receiving 6 miRNA molecules as inputs. In the first cascade these inputs interact with translating molecules releasing output strands that are used as inputs in the next cascade. To minimize non-specific interactions between the circuits, computational optimization means were used. Their system operated successfully and autonomously in the presence of

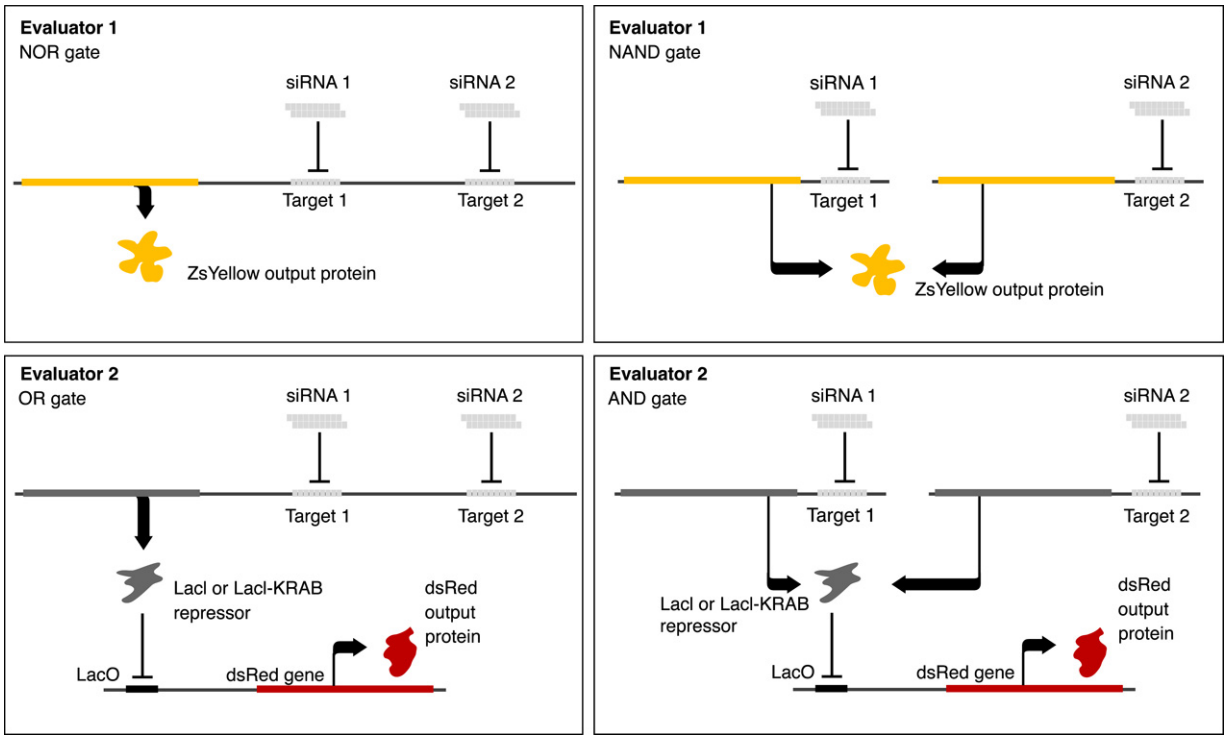

Fig. 5. Molecular system for the evaluation of logic gates NOR, NAND, OR and AND. Two variables gates are shown. Synthetic siRNAs are used as input and expression levels of the fluorescent protein as output. Evaluators 1 and 2 are formed by inserting siRNAs targets into non-coding regions of one (for NOR and OR gates) or two (for NAND and AND gates) synthetic mRNAs, encoding for ZsYelllow protein (for Evaluator 1, NOR and NAND gates) or a repressor (LacI or LacI-KRAB), which suppresses the expression of dsRed protein (for Evaluator 2, OR and AND gates).

mouse brain total RNA extract, an environment that supposedly simulates the conditions existing in living cells.

The simplicity, modularity and scalability of the system demonstrated by Winfree and colleagues enable a promising foundation for future applications.

7. Evaluation of a specified combination of synthetic small interfering RNAs, *in vivo*

Benenson and colleagues developed, in living cells, a molecular system for the evaluation of logic expressions over the presence (or absence) of siRNA [36]. Synthetic small interfering RNAs (siRNAs; a class of 20–25 nucleotides long double-stranded RNA molecules that inhibit mRNA translation to protein) were used as input and the expression of a fluorescent protein was used as the output. Target sequences of siRNAs were consecutively fused into non-coding regions (UTR) of a synthetic mRNA molecule that encodes for the output or for its repressing protein which inhibits translation upon the binding of siRNA to its specific target sequence on the mRNA. The cells were genetically engineered to possess these mRNA molecules. Different combinations of siRNA molecules were inserted into the cells by transfection (a method to transfer foreign DNA molecules into the cells by making small holes in their membrane; this can be done either by electrical or chemical means).

They implemented successfully two types of evaluator systems (Fig. 5): (1) siRNAs capable of regulating directly the reporting mRNA that encodes for a fluorescent protein (ZsYelllow). This evaluator realizes NAND and NOR gates using siRNAs as input and (2) siRNAs capable of regulating the repressor (LacI or LacI-KRAB) that regulates the reporter mRNA that encodes for another fluorescent protein (dsRed). This second evaluator realizes AND and OR gates using siRNAs as input. They implemented molecular circuits with up to five logic variables, confirming the computation with all the possible combinations of the siRNAs, which simulates all the variables combinations.

In future, they plan to develop a sensing module using intracellular mRNAs as input. This will allow implementation of Boolean expression of mRNA species (such as a conjunctive normal form, CNF, or a disjunctive normal form, DNF). The synthetic siRNAs, upon insertion into living cells and interaction with intracellular mRNA molecules, will serve as translator molecules. The development of a sensing module, would allow arbitrary Boolean decision-making, using endogenous mRNA species as inputs.

8. Future directions of biomolecular computers

Since molecular computers can directly access data encoded in intracellular bio-molecules, in a way electronic computers will never do, we believe that this new computer species is of fundamental importance and will be proved to be valuable for a wide range of biotechnological and biomedical applications.

Development of a molecular computer that conducts computations in living cells has to cross few barriers: (1) delivery of its hardware and software into living cells;

(2) interference of cellular components, such as ions and enzymes in computer's activity; (3) the hardware and software should not be toxic to the cells; (4) computation should be sufficiently fast to overcome the computer's degradation; (5) the molecular computer should interact with cell molecules in their physiological concentration. This should be correlated with the delivery method to ensure that the molecular computer's components will be in adequate concentrations and (6) computation in the cells should preferably take place in the locality of the intracellular molecules that serve as input. Overcoming all those barriers will not be simple, but doing so holds great promise for both analysis applications in biological systems and therapeutic applications in medicine.

Acknowledgement

We thank K. Katzav for the prompt and excellent preparation and design of figures.

References

[1] L.M. Turing, On computable numbers, with an application to the entcheidungsproblem, Proc. Lond. Math. Soc. 42 (1936) 230–265.
[2] C.H. Bennett, The thermodynamics of computation — Review, Int. J. Theoret. Phys. 21 (1982) 905–940.
[3] A.M. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (1994) 1021–1024.
[4] Q. Ouyang, et al., DNA solution of the maximal clique problem, Science 278 (1997) 446–449.
[5] R.J. Lipton, DNA solution of hard computational problems, Science 268 (1995) 542–545.
[6] R.S. Braich, et al., Solution of a 20-variable 3-SAT problem on a DNA computer, Science 296 (2002) 499–502.
[7] Liu Q., et al., DNA computing on surfaces, Nature 403 (2000) 175–179.
[8] D. Faulhammer, et al., Molecular computation: RNA solutions to chess problems, Proc. Natl. Acad. Sci. USA 97 (2000) 1385–1389.
[9] C. Mao, et al., Logical computation using algorithmic self-assembly of DNA triple-crossover molecules, Nature 407 (2000) 493–496.
[10] A.J. Ruben, et al., The past, present and future of molecular computing, Nat. Rev. Mol. Cell. Biol. 1 (2000) 69–72.
[11] J. Von Neumann, First draft of a report on EDVAC, 1945.
[12] Stojanovic M.N., et al., A deoxyribozyme-based molecular automaton, Nat Biotechnol. 21 (2003) 1069–1074.
[13] Y. Benenson, et al., Programmable and autonomous computing machine made of biomolecules, Nature 414 (2001) 430–434.
[14] P. Rothemund, A DNA and restriction enzyme implementation of Turing machines, DNA based computers II, in: Proc. Second DIMACS Workshop on DNA-Based Computers, 1995.
[15] Y. Benenson, et al., DNA molecule provides a computing machine with both data and fuel, Proc. Natl. Acad. Sci. USA 100 (2003) 2191–2196.
[16] R. Adar, et al., Stochastic computing with biomolecular automata, Proc. Natl. Acad. Sci. USA 101 (2004) 9960–9965.
[17] Y. Benenson, et al., An autonomous molecular computer for logical control of gene expression, Nature 429 (2004) 423–429.
[18] E. Shapiro, et al., Bringing DNA computers to life, Sci. Amer. 294 (2006) 44–51.
[19] E. Shapiro, A mechanical turing machine: Blueprint for a biomolecular computer. in: 5th International Meeting on DNA Based Computers, 1999.
[20] X.J. Feng, S. Hooshangi, D. Chen, G. Li, R. Weiss, H. Rabitz, Optimizing genetic circuits by global sensitivity analysis, Biophys. J. 87 (2004) 2195–2202.
[21] A. Becskei, L. Serrano, Engineering stability in gene networks by autoregulation, Nature 405 (2000) 590–593.
[22] A. Becskei, B. Seraphin, L. Serrano, Positive feedback in eukaryotic gene networks: Cell differentiation by graded to binary response conversion, EMBO J. 20 (2001) 2528–2535.
[23] F.J. Isaacs, J. Hasty, C.R. Cantor, J.J. Collins, Prediction and measurement of an autoregulatory genetic module, Proc. Natl. Acad. Sci. USA 100 (2003) 7714–7719.
[24] T.S. Gardner, C.R. Cantor, J. Collins, Construction of a genetic toggle switch in Escherichia coli, Nature 403 (2000) 339–342.
[25] M.R. Atkinson, M.A. Savageau, J.T. Myers, A.J. Ninfa, Development of genetic circuitry exhibiting toggle switch or oscillatory behavior in Escherichia coli, Cell 113 (2003) 597–607.
[26] M.B. Elowitz, S. Leibler, A synthetic oscillatory network of transcriptional regulators, Nature 403 (2000) 335–338.
[27] N. Barkai, S. Leibler, Biological rhythms: Circadian clocks limited by noise, Nature 403 (2000) 267–268.
[28] J.M.G. Vilar, H.Y. Kueh, N. Barkai, S. Leibler, Mechanisms of noise-resistance in genetic oscillators, Proc. Natl. Acad. Sci. USA 99 (2002) 5988–5992.
[29] T. Bulter, S.G. Lee, W.W. Wong, E. Fung, M.R. Connor, J.C. Liao, Design of artificial cell–cell communication using gene and metabolic networks, Proc. Natl. Acad. Sci. USA 101 (2004) 2299–2304.
[30] W.J. Blake, F.J. Isaacs, Synthetic biology evolves, Trends Biotechnol. 22 (2004) 321–324.
[31] J.Z. Hilt, Nanotechnology and biomimetic methods in therapeutics: Molecular scale control with some help from nature, Adv. Drug Deliv. Rev. 56 (2004) 1533–1536.
[32] T. Kaczorowski, et al., Purification and characterization of the Fok I restriction endonuclease, Gene 80 (1989) 209–216.
[33] M. Soreni, et al., Parallel biomolecular computation on surfaces with advanced finite automata, J. Am. Chem. Soc 127 (2005) 3935–3943.
[34] C. Capoulade, et al., Apoptosis of tumoral and nontumoral lymphoid cells is induced by both mdm2 and p53 antisense oligodeoxynucleotides, Blood 97 (2001) 1043–1049.
[35] G. Seelig, et al., Enzyme-free nucleic acid logic circuits, Science 314 (2006) 1585–1588.
[36] K. Rinaudo, et al., A universal RNAi-based logic evaluator that operates in mammalian cells, Nat. Biotechnol. 25 (2007) 795–801.