nature nanotechnology

LETTERS

PUBLISHED ONLINE: 2 AUGUST 2009 | DOI: 10.1038/NNANO.2009.203

Molecular implementation of simple logic programs

Tom Ran[1], Shai Kaplan[2,3] and Ehud Shapiro[1,2]*

Autonomous programmable computing devices made of biomolecules could interact with a biological environment and be used in future biological and medical applications[1–7]. Biomolecular implementations of finite automata[8,9] and logic gates[4,10–13] have already been developed[14–18]. Here, we report an autonomous programmable molecular system based on the manipulation of DNA strands that is capable of performing simple logical deductions. Using molecular representations of facts such as *Man*(*Socrates*) and rules such as *Mortal*(*X*) ← *Man*(*X*) (Every Man is Mortal), the system can answer molecular queries such as *Mortal*(*Socrates*)*?* (Is Socrates Mortal?) and *Mortal*(*X*)*?* (Who is Mortal?). This biomolecular computing system compares favourably with previous approaches in terms of expressive power, performance and precision[2,4,8,9,11,12,19]. A compiler translates facts, rules and queries into their molecular representations and subsequently operates a robotic system that assembles the logical deductions and delivers the result. This prototype is the first simple programming language with a molecular-scale implementation.

Our logic system consists of propositions, implications and queries, represented by short DNA molecules, some single-stranded and some double-stranded, as shown in Fig. 1a. A proposition *p*, such as 'Socrates is a Man', is represented by a double-stranded (ds) DNA molecule with a sticky end representing *p* on one end and a fluorophore (a fluorescent molecule) with a matching quencher (a molecule that absorbs excitation energy from a fluorophore) at its other end. A query *p?*, such as 'Is Socrates a Man?', is represented by a dsDNA molecule with a sticky end complementary to the representation of *p* and a recognition site (marked light red) for the restriction enzyme *Fok*I. A molecular query *p?* can be positively answered by the molecular proposition *p*, because they have complementary sticky ends, as shown in Fig. 1b. The sticky end of the query molecule *q?* hybridizes with that of the proposition molecule *q*. *Fok*I is attracted to its recognition site on the query (before hybridization or possibly following it) and cleaves the proposition molecule *q*. This results in the separation of its remaining sense and antisense strands, which in turn abolishes the quenching of the green fluorophore, allowing the fluorophore to emit green light in response to light excitation. This green emission can be interpreted by an outside observer as a positive response to the query.

An implication $p \leftarrow q$ (for example 'Socrates is Mortal if Socrates is a Man') is represented by a hairpin single-stranded (ss) DNA with three components: a sticky end representing the proposition *p*, a segment complementary to the representation of *q* and a segment that together with an auxiliary complementary strand forms a recognition site for *Fok*I. The implication $p \leftarrow q$ can be used to reduce the query *p?* to the query *q?*, meaning that to positively answer *p?* it suffices to positively answer *q?* This query reduction is justified by the deduction rule *modus ponens*, which states that from *q* and the implication $p \leftarrow q$ one can deduce *p*. The reduction, shown in Fig. 1c, proceeds at the molecular level as follows: the sticky end of the query molecule *p?* hybridizes with that of the implication molecule $p \leftarrow q$, *Fok*I is attracted to its recognition site in the query and cleaves the implication molecule, releasing a ssDNA that hybridizes with the auxiliary strand to form the new query molecule *q?*.

We illustrate the richness of the molecular logic system by using it to implement simple logic programs. Logic programming[20,21] is an approach to computer programming that uses a subset of first-order logic as a programming language. Logic programming has applications in artificial intelligence research[22], natural language processing[23,24] and concurrent programming[25]. Logic programming inspired several designs for DNA-based inference systems[14,15,17,18], including an example of laboratory-scale DNA computing with a human-assisted protocol[16]. These implementations of propositional logic programs were considered a stepping stone to predicate logic, which has the computation power of a Turing machine[26,27]. Here, we focus on logic programs with unary predicates and simple implications, examples of which are shown in Fig. 2a.

The facts depicted in Fig. 2a, such as *Man*(*Socrates*) and *Philosopher*(*Plato*), are a means of stating that certain properties (for example, being a Man or being a Philosopher) hold for certain individuals (for example, Socrates, Plato). Universal rules of the form $Mortal(X) \leftarrow Man(X)$ state that certain facts (for example, *Man*(*Socrates*)) imply other facts (for example, *Mortal*(*Socrates*)). Such a universal rule can be thought of as a shorthand for all its instances, including $Mortal(Socrates) \leftarrow Man(Socrates)$ and $Mortal(Plato) \leftarrow Man(Plato)$. Queries are also shown in Fig. 2a. A query of the form *Man*(*Plato*)*?* asks whether the individual Plato has the property of being a Man. The molecular representation of such facts, rules and queries is shown in Fig. 2a, and is derived from their translation to propositional logic, and the molecular representation of propositional logic discussed above. Logic program facts such as *Man*(*Socrates*) are translated to propositional implications 'Man(X)' ← 'X = Socrates', logic program rules are translated to their corresponding implications, and logic program queries such as *Mortal*(*Socrates*)*?* are translated to a combination of the propositional assumption 'X = Socrates' and the query 'Mortal(X)'? Using this representation it is possible to encode facts with the same property but different individuals, such as *Man*(*Plato*) and *Man*(*Achilles*), as well as facts with different properties but the same individual, such as *Man*(*Plato*) and *Philosopher*(*Plato*).

A logic program is a finite set of facts and rules. A query is answered by a logic program by determining whether the query is a logical consequence of the program. Logical consequences are obtained by applying deduction rules, and our basic step of logical deduction is query reduction. An example of answering the query *Man*(*Plato*)*?* with the fact *Man*(*Plato*) is shown in full molecular-level detail in Fig. 2b. The query 'Man(X)'? is first reduced to the query 'X = Plato'?, which is then answered positively using the assumption 'X = Plato', previously supplied as part of the

[1]Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel, [2]Department of Biological Chemistry, Weizmann Institute of Science, Rehovot 76100, Israel, [3]Department of Molecular Cell Biology, Weizmann Institute of Science, Rehovot 76100, Israel.
*e-mail: ehud.shapiro@weizmann.ac.il

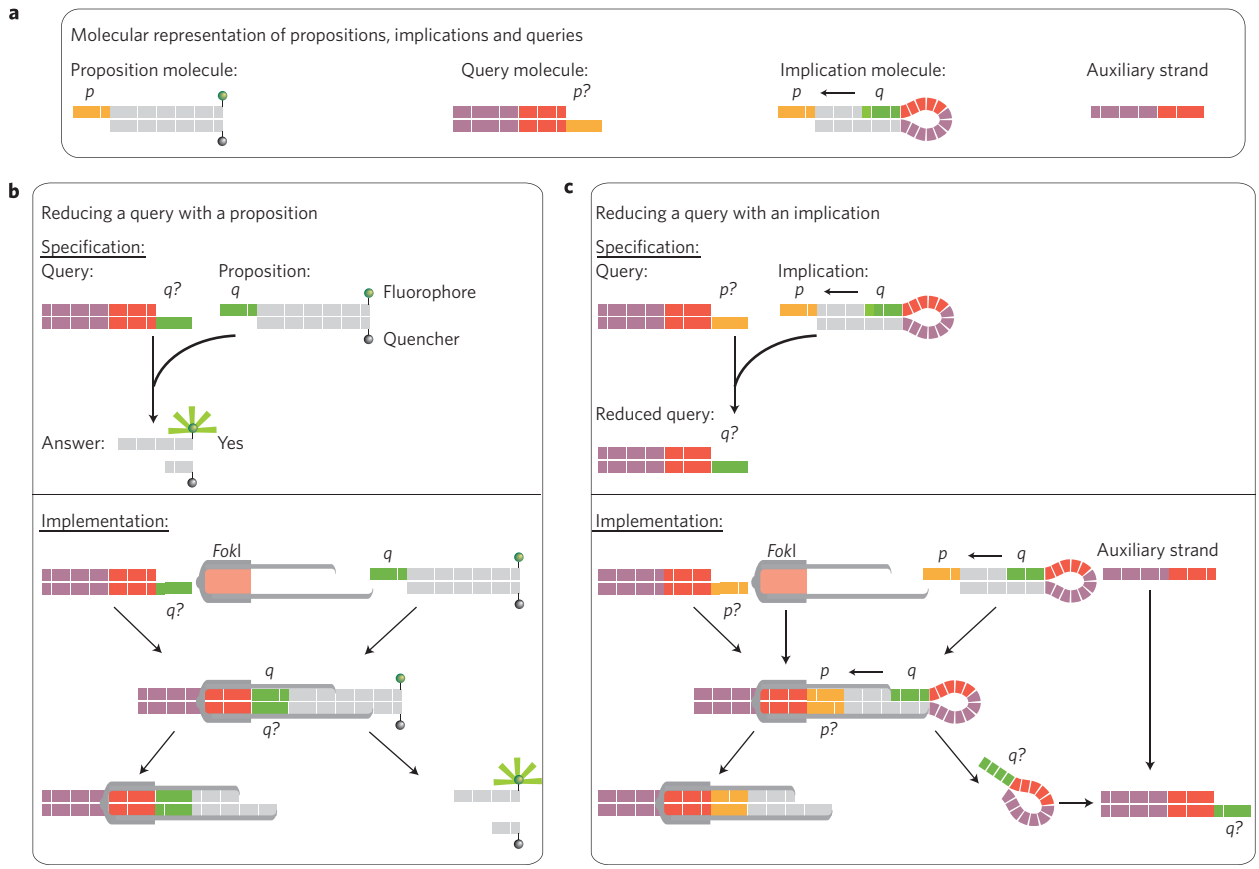


Figure 1 | Molecular implementation of propositional logic statements and deductions. a, A proposition *p* is represented by a dsDNA molecule with a 4-nucleotide sticky end. A query *p?* is represented by a dsDNA molecule with two components: a 5-nucleotide recognition site (marked red) for the restriction enzyme *FokI* and a 4-nucleotide sticky end that is complementary to the representation of *p*. An implication $p \leftarrow q$ is represented by a hairpin ssDNA with three components: a 4-nucleotide sticky end representing the proposition *p*, a 4-nucleotide section complementary to the representation of *q* and a 5-nucleotide section, that together with a corresponding auxiliary strand, forms a recognition site for *FokI*. **b**, The proposition *q* can be used to answer the query *q?* positively as follows: the query molecule *q?* hybridizes with the proposition molecule *q* and attracts *FokI* to its recognition site, which in turn cleaves the molecule *q*, causing the separation of the remaining sense and antisense strands, which in turn abolishes the quenching of the green fluorophore by the quencher, causing the fluorophore to emit green light in response to light excitation. This green emission can be interpreted by an outside observer as a positive answer to the query. **c**, The implication $p \leftarrow q$ can be used to reduce the query *p?* to the query *q?*. The reduction is justified by the deduction rule *modus ponens*, which states that from *q* and $p \leftarrow q$ one can deduce *p*. The sticky ends of the molecules *p?* and $p \leftarrow q$ hybridize, *FokI* is attracted to its recognition site in the query molecule and cleaves the implication molecule $p \leftarrow q$, releasing a ssDNA that, together with the auxiliary strand, forms the new query molecule *q?*.

query. The final results of 12 biochemical reactions of various combinations of queries and facts are depicted in Fig. 2c. The dynamics of these reactions is shown in Fig. 2d.

A more elaborate deduction is shown in Fig. 3, with the molecular details comprising 20 basic bimolecular steps omitted (a complete illustration of the molecular interactions of this deduction is given in Supplementary Fig. S1). The molecular query *Mortal*(*Plato*)*?* is answered positively from a molecular logic program containing molecules representing the rules *Mortal*(*X*) ← *Man*(*X*), *Man*(*X*) ← *Greek*(*X*), and the fact *Greek*(*Plato*), as well as spurious rules and facts, as shown in Fig. 3b. The major logical steps of the molecular deduction process are shown in Fig. 3a. The results of molecular deductions of various logic programs with various queries are shown in Fig. 3b, attesting to the robustness of the molecular implementation. The kinetics of the deduction process is shown in Fig. 3c. Such a cascading of computational steps is possible only when the inputs and outputs of each step are of the same type. Cascading of molecular computations has previously been shown by Seelig and colleagues[4]. Their system consists of DNA logic gates that require hours per logic gate operation, and the time per operation increases when operations are cascaded. In contrast, our system has a half time of 3 min per logical deduction, and deductions do not slow down measurably when cascaded. In addition, our molecular implementation maintains efficient digital behaviour, exhibiting very low signal leakage and keeping a high signal yield and strength when cascaded, eliminating the need for signal restoration thresholds and amplifier gates to protect against noise, signal loss and leaky reactions as needed in earlier systems[4,12]. To demonstrate these features of our system, we designed a simple four-step cascaded deduction experiment. The computation completes with a half time of ~2 min, or 30 s per deduction (see Supplementary Fig. S2). Moreover, the cascading signal does not slow down significantly or decay during the deduction process. Our system is designed so that each query molecule has the potential to cleave indefinitely its corresponding rule molecules without undergoing any change and without requiring an external energy supply[28], and therefore a small percentage of the cleaving molecules can cleave almost all of their target molecules.

A query such as *Mortal*(*Plato*)*?* asks about a specific individual, in this case Plato, and the answer can either be '*Yes*' or '*No*'. More generally, the existential query *Mortal*(*X*)*?* asks *Who is Mortal?*

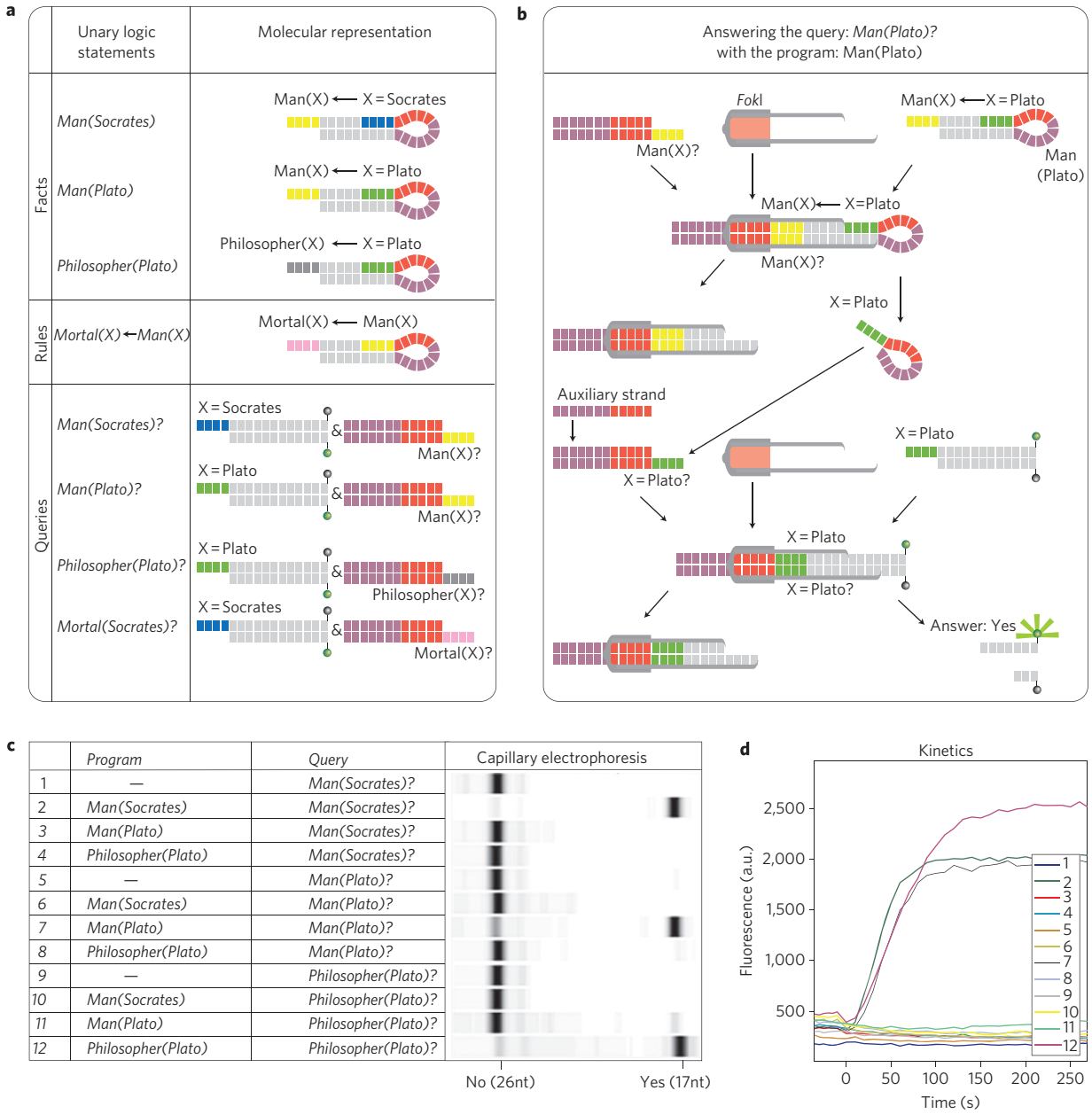
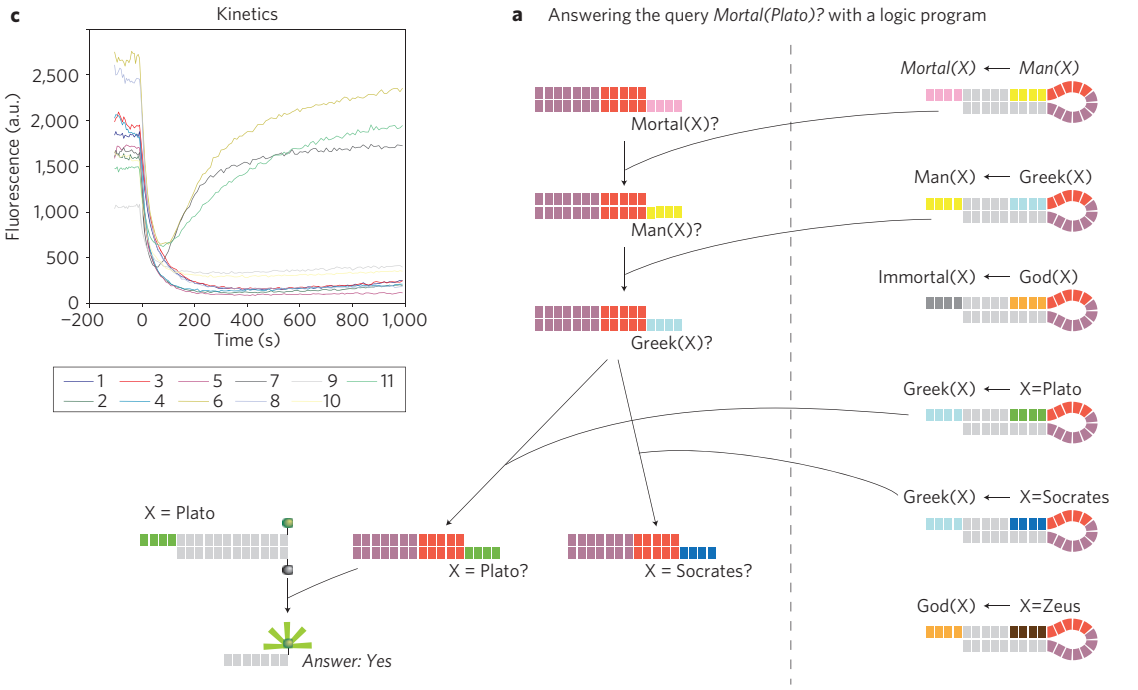


Figure 2 | Molecular implementation of simple logic programs. a, Facts, rules and queries and their molecular implementation. Facts such as *Man(Plato)* (Plato is a Man) are implemented by the propositional implication 'Man(X)' ← 'X = Plato', logic program rules such as *Mortal(X)* ← *Man(X)* are implemented by the corresponding implications, and logic program queries such as *Mortal(Plato)?* are implemented by adding as assumption the proposition 'X = Plato' and making the propositional query 'Mortal(X)'?, as further explained and justified in the text. **b**, Answering the logic program query *Man(X)?* using the logic program in **a**. The query 'Man(X)'? is first reduced to the query 'X = Plato'?, which is positively answered by the corresponding proposition previously added as an assumption. **c**, Each molecular logic program was queried as depicted in the table. Each lane of the capillary gel electrophoresis presents the correct expected '*No*' or '*Yes*' answer (an uncleaved 26 nucleotide DNA strand or a cleaved 17 nucleotide DNA strand, respectively). **d**, Kinetics of the deduction process showing that answers are deduced in about one minute.

(more precisely, is there an X that is Mortal?), and ideally should be answered with the list of individuals who are Mortal, if there are any, or '*No*' otherwise. This type of query is an exception that uses a single variable that can receive multiple bindings. We implement existential queries by adding to the propositional query 'Mortal(X)'? several assumptions, 'X = Plato', 'X = Socrates', and so on, one for each relevant individual. In order to know which individual participated in successful deductions at the molecular level we tag each such assumption molecule with a different unique fluorescent colour. Thus, for each individual that forms an answer to such an existential query we observe a rise in fluorescent intensity in its associated colour. This is demonstrated by running the molecular logic program depicted in Fig. 4 against the queries 'Who is suitable for the army?' and 'Who is suitable for the academy?' The computation is depicted in Fig. 4 and involves more than 70 different molecular species (this deduction is shown in full molecular detail in Supplementary Fig. S3).

Query reduction works top down from rules to facts. An alternative deduction method works bottom up, deducing new facts from known facts and checking from time to time if a fact corresponding to the query is deduced. Our system is also capable of bottom-up deduction, as shown in Fig. 5. Our bottom-up system is not



b

	Program	Query	Capillary electrophoresis
1	*Man(X) ← Greek(X), Greek(Socrates)*	*Mortal(Aristo)?*	
2	*Man(X) ← Greek(X), Greek(Plato)*	*Mortal(Plato)?*	
3	*Mortal(X) ← Man(X), Greek(Socrates)*	*Mortal(Aristo)?*	
4	*Mortal(X) ← Man(X), Greek(Plato)*	*Mortal(Plato)?*	
5	*Mortal(X) ← Man(X), Man(X) ← Greek(X), Greek (Plato)*	*Mortal(Aristo)?*	
6	*Mortal(X) ← Man(X), Man(X) ← Greek(X), Greek (Plato)*	*Mortal(Plato)?*	
7	*Mortal(X) ← Man(X), Man(X) ← Greek(X), Greek (Socrates)*	*Mortal(Aristo)?*	
8	*Mortal(X) ← Man(X), Man(X) ← Greek(X), Greek(Socrates)*	*Mortal(Plato)?*	
9	*Mortal(X) ← Man(X), Man(X) ← Greek(X), God(Zeus)*	*Mortal(Zeus)?*	
10	*God(Zeus)*	*Immortal(Zeus)?*	
11	*Immortal(X) ← God(X), God(Zeus)*	*Immortal(Zeus)?*	

No (26nt) Yes (17nt)

Figure 3 | Long deductions. a, Answering whether Plato is Mortal, showing fewer molecular details compared to Fig. 2b. **b**, Multiple logic programs and queries were tested. When one of the required rules is absent (lanes 1, 2, 3, 4, 10) or one of the required fact molecules is absent (lanes 5, 8, 9) the answer is '*No*'. When all rule molecules and the required fact molecule are present the answer is '*Yes*' (lanes 6, 7, 11). **c**, Kinetics of the deduction process.

restricted to rules with a single condition and allows conjunction of conditions in a rule, and as such is logically more powerful than our top-down implementation. Our example, adapted from Maslow's hierarchy of needs[29], includes the rule that a person is happy if he or she has love and money. We tested a program comprised of the rule *In_love*(*X*) & *Has_Money*(*X*)→*Happy*(*X*) and several combinations of the facts; *Has_Money*(*Maslow*), *Has_Money*(*Freud*), *In_love*(*Maslow*) and *In_love*(*Fromm*). Only when the logic program included all the factual conditions for Maslow to be happy was the answer '*Yes*'. Other logic programs, which lacked one of the facts required in order for the individual queried about to be happy, resulted in negative answers, as shown in Fig. 5. Using bottom-up deductions, our system design is complete, in principle, for unary predicate logic with no function symbols (that is, it can deduce any logical consequence of the program), up to a certain limited number of predicate and constant symbols. The practical limitation is the ability to encode the needed number of predicate and constant symbols with the available short sequences allowed by our design.

A key reason for choosing logic programming as a basis for biomolecular computing is its clear semantics, allowing its use as a high-level programming language. A high-level language allows a more abstract, intuitive and thus easier to use way of programming[30]. In the early computer days, programs were written in executable low-level machine or assembly languages. Later, compilers were introduced, enabling translation of high-level language programs into their corresponding executable programs. We developed a hybrid *in silico*/*in vitro* system that supports the creation and execution of molecular logic programs in a fashion similar to electronic computers (see Supplementary Fig. S4); as in electronic computers, programming is done in a high-level language and then translated by a compiler to create an executable program. All experiments described in this work were executed by this system.

Using this system one can specify logic programs with facts and rules as well as various queries as a text file. A compiler translates this logic program text into a molecular representation according to the translations described above, by associating each symbol in the program, that is, property or individual, with a unique 4-nucleotide sticky end of a DNA molecule from a predefined 4-nucleotide sticky library. The output of the compiler is the list of molecules that constitute the molecular implementation of the logic program as well as a robot control program that assembles these 'software'

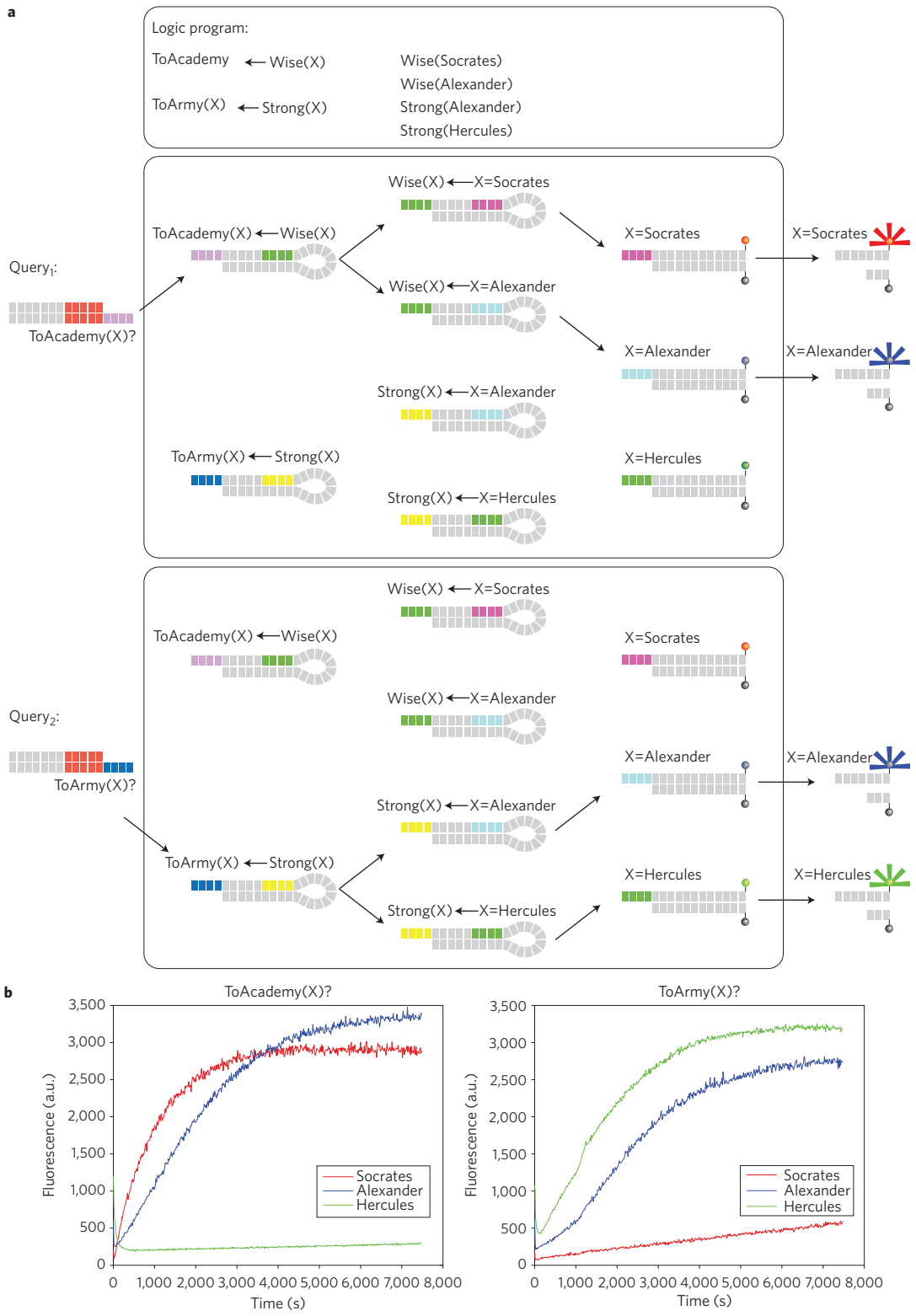
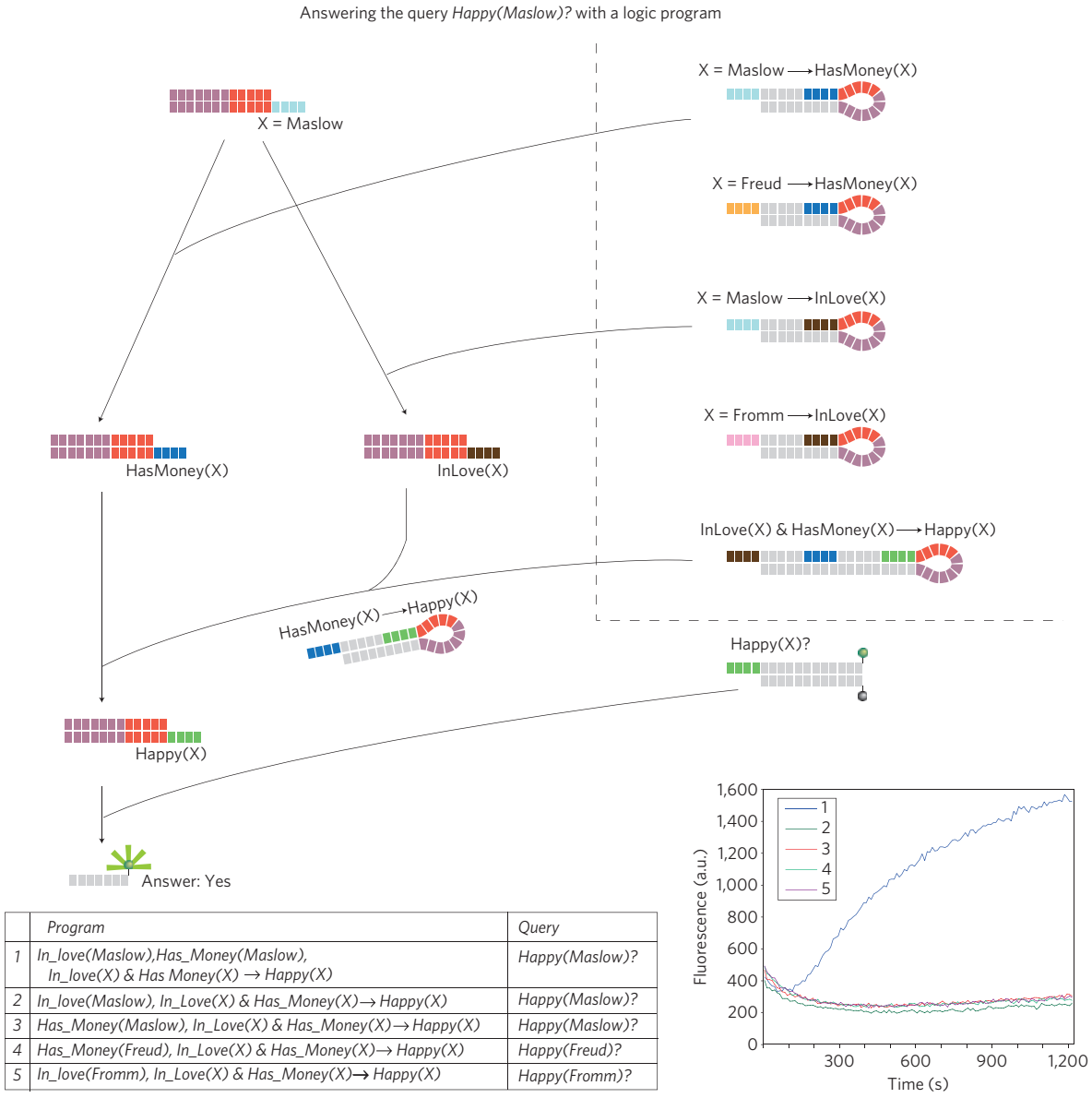


Figure 4 | 'Who is suitable for the army?' and 'Who is suitable for the academy?'. a, By adding several individual assumption molecules, each tagged with a fluorescent colour, each molecularly derived individual answer would result in a fluorescent intensity rise in the individual's colour. **b**, When the depicted program was queried *ToArmy(X)?* the correct answers *Socrates* and *Alexander* were obtained and when queried *ToAcademy(X)?* the correct answers *Alexander* and *Hercules* were obtained.

molecules together with the 'hardware' enzyme and buffer into one dedicated well. Multiple logic programs can be compiled and executed simultaneously, each in a dedicated well. Following the biochemical reaction assembly, the 96-well plate is inserted into a microplate reader that reads the fluorescent output, followed by a computational signal analysis that provides the query's answer. Essentially, if the solution in a well of a plate in which a molecular deduction took place 'glows', then the answer for this deduction is



	Program	Query
1	*In_love(Maslow),Has_Money(Maslow), In_love(X) & Has Money(X) → Happy(X)*	*Happy(Maslow)?*
2	*In_love(Maslow), In_Love(X) & Has_Money(X)→ Happy(X)*	*Happy(Maslow)?*
3	*Has_Money(Maslow), In_Love(X) & Has_Money(X)→ Happy(X)*	*Happy(Maslow)?*
4	*Has_Money(Freud), In_Love(X) & Has_Money(X)→ Happy(X)*	*Happy(Freud)?*
5	*In_love(Fromm), In_Love(X) & Has_Money(X)→ Happy(X)*	*Happy(Fromm)?*

Figure 5 | Bottom-up deductions. A person is happy if he or she has love and money (example adapted from Maslow's hierarchy of needs[29]). Only when the logic program includes all the factual conditions for Maslow to be happy was the answer '*Yes*' (program 1). Logic programs lacking one of the facts required for the individual to be happy resulted in negative answers (programs 2–5).

'*Yes*'. This automated system provides a powerful tool to program and execute molecular computations on a short timescale measured in minutes. Our work demonstrates molecular execution of simple logic programs based on specific coding of DNA strands and their molecular manipulation.

Received 30 March 2009; accepted 1 July 2009; published online 2 August 2009

References

1. Gardner, T. S., Cantor, C. R. & Collins, J. J. Construction of a genetic toggle switch in *Escherichia coli*. *Nature* **403,** 339–342 (2000).
2. Benenson, Y., Gil, B., Ben-Dor, U., Adar, R. & Shapiro, E. An autonomous molecular computer for logical control of gene expression. *Nature* **429,** 423–429 (2004).
3. Basu, S., Gerchman, Y., Collins, C. H., Arnold, F. H. & Weiss, R. A synthetic multicellular system for programmed pattern formation. *Nature* **434,** 1130–1134 (2005).
4. Seelig, G., Soloveichik, D., Zhang, D. Y. & Winfree, E. Enzyme-free nucleic acid logic circuits. *Science* **314,** 1585–1588 (2006).
5. Rinaudo, K. *et al.* A universal RNAi-based logic evaluator that operates in mammalian cells. *Nature Biotechnol.* **25,** 795–801 (2007).
6. Deans, T. L., Cantor, C. R. & Collins, J. J. A tunable genetic switch based on RNAi and repressor proteins for regulating gene expression in mammalian cells. *Cell* **130,** 363–372 (2007).
7. Kobayashi, H. *et al.* Programmable cells: interfacing natural and engineered gene networks. *Proc. Natl Acad. Sci. USA* **101,** 8414–8419 (2004).
8. Benenson, Y. *et al.* Programmable and autonomous computing machine made of biomolecules. *Nature* **414,** 430–434 (2001).
9. Stojanovic, M. N. & Stefanovic, D. A deoxyribozyme-based molecular automaton. *Nature Biotechnol.* **21,** 1069–1074 (2003).
10. Breaker, R. R. Engineered allosteric ribozymes as biosensor components. *Curr. Opin. Biotechnol.* 31–39 (2002).
11. Macdonald, J. *et al.* Medium scale integration of molecular logic gates in an automaton. *Nano Lett.* **6,** 2598–2603 (2006).
12. Zhang, D. Y., Turberfield, A. J., Yurke, B. & Winfree, E. Engineering entropy-driven reactions and networks catalyzed by DNA. *Science* **318,** 1121–1125 (2007).
13. Isaacs, F. J. *et al.* Engineered riboregulators enable post-transcriptional control of gene expression. *Nature Biotechnol.* **22,** 841–847 (2004).
14. Kobayashi, S., Yokomori, T., Sampei, G. & Mizobuchi, K. DNA implementation of simple horn clause computation. *IEEE International Conference on Evolutionary Computation*, 213–217 (Indianapolis, USA, 1997).

15. Kobayashi, S. Horn clause computation with DNA molecules. *J. Comb. Optim.* **3,** 277–299 (1999).
16. Siewicz, P., Janczak, T., Mulawaka, J. J. & Plucienniczak, A. The inference via DNA computing. *Congress on Evolutionary Computation*, 988–993 (Washington, USA, 1999).
17. Siewicz, P., Janczak, T., Mulawaka, J. J. & Plucienniczak, A. The inference based on molecular computing. *Int. J. Cybern. Syst.* **31/3,** 283–315 (2000).
18. Uejima, H., Hagiya, M. & Kobayashi, S. Horn clause computation by self-assembly of DNA molecules. *Lecture Notes in Computer Science, DNA Computing* **7,** 308–320 (2002).
19. Adar, R. *et al*. Stochastic computing with biomolecular automata. *Proc. Natl Acad. Sci. USA* **101,** 9960–9965 (2004).
20. Sterling, L. & Shapiro, E. *The Art of Prolog: Advanced Programming Techniques* (MIT Press, 1986, 1994).
21. Lloyd, J. W. *Foundations of Logic Programming* (Springer-Verlag, 1987).
22. Bratko, I. *Prolog Programming for Artificial Intelligence* (Addison-Wesley, 2001).
23. Covington, M. A. *Natural Language Processing for Prolog Programmers* (Prentice Hall, 1994).
24. Gazdar, G. & Mellish, C. S. *Natural Language Processing in Prolog: an Introduction to Computational Linguistics* (Addison-Wesley, 1989).
25. Shapiro, E. *Concurrent Prolog: Collected Papers*, Vols 1 and 2 (MIT Press, 1987).
26. Tarnland, S. A. Horn clause computability. *J. Comb. Optim.* **17,** 215–226 (1977).
27. Shapiro, E. Alternation and the computational complexity of logic programs. *J. Logic Programming* **1,** 19–33 (1984).
28. Benenson, Y., Adar, R., Paz-Elizur, T., Livneh, Z. & Shapiro, E. DNA molecule provides a computing machine with both data and fuel. *Proc. Natl Acad. Sci. USA* **100,** 2191–2196 (2003).
29. Maslow, A. A theory of human motivation. *Psychol. Rev.* **50,** 370–396 (1943).
30. Winfree, E. Biochemical logic: submerged circuits of floating DNA. *ENGenious*, 52–54 (2007).

Acknowledgements

We thank R. Adar, M. Kahan and B. Gil for their assistance and advice and A. Mishali and G. Brodsky for the preparation of the figures. This research was supported by The Israel Science Foundation grant no. 285/02, a research grant from the Clore Center for Biological Physics, a research grant from the Louis Chor Memorial Trust, a research grant from the Estate of Funnie Sherr, the Cymerman-Jubskind Prize, the Estate of Karl Felix Jakubskind and by the European Union FP7-ERC-AdG. S.K. is supported by the Yeshaya Horowitz association through the Center for Complexity Science. E.S. is the Incumbent of the Harry Weinrebe Professorial Chair of Computer Science and Biology and of the France Telecom–Orange Excellence Chair for Interdisciplinary Studies of the Paris 'Centre de Recherche Interdisciplinaire' (FTO/CRI).

Author contributions

E.S. led the project. T.R. conceived the biomolecular design and the compiler and designed and performed the experiments. T.R. and S.K. wrote the analysis tools and automated the experimental protocols. T.R. and E.S. wrote the paper.

Additional information

Supplementary information accompanies this paper at www.nature.com/naturenanotechnology. Reprints and permission information is available online at http://npg.nature.com/reprintsandpermissions/. Correspondence and requests for materials should be addressed to E.S.