Acta Biotheor
DOI 10.1007/s10441-015-9258-8

REGULAR ARTICLE

Computing with Synthetic Protocells

Alexis Courbet[2] · Franck Molina[2] · Patrick Amar[1]

Received: 5 February 2015 / Accepted: 5 May 2015

Abstract In this article we present a new kind of computing device that uses biochemical reactions networks as building blocks to implement logic gates. The architecture of a computing machine relies on these generic and composable building blocks, *computation units*, that can be used in multiple instances to perform complex boolean functions. Standard logical operations are implemented by biochemical networks, encapsulated and insulated within synthetic vesicles called protocells. These protocells are capable of exchanging energy and information with each other through transmembrane electron transfer. In the paradigm of computation we propose, *protoputing*, a machine can solve only one problem and therefore has to be built specifically. Thus, the programming phase in the standard computing paradigm is represented in our approach by the set of assembly instructions (specific attachments) that directs the wiring of the protocells that constitute the machine itself. To demonstrate the computing power of protocellular machines, we apply it to solve a NP-complete problem, known to be very demanding in computing power, the 3-SAT problem. We show how to program the assembly of a machine that can verify the satisfiability of a given boolean formula. Then we show how to use the massive parallelism of these machines to verify in less than 20 min *all* the valuations of the input variables and output a fluorescent signal when the formula is satisfiable or no signal at all otherwise.

Keywords Synthetic biology · Biocomputing · 3-SAT · Protocell · Protoputing

✉ Patrick Amar
pa@lri.fr

[1] LRI, Université Paris Sud - UMR CNRS 8623, Bât. 650, 91405 Orsay Cedex, France

[2] Sys2diag, FRE CNRS 3690, 1682 rue de la Valsière, 34184 Montpellier, France

Published online: 13 May 2015

1 Introduction

What is Computation? One definition could be "the goal-oriented process that transforms a representation of input information into a representation of output information". The process itself can be *iterative* (or in another form *recursive*), in this case it is called an *algorithm*, but other forms of processing can be used, such as: *neural networks* or *first order logic*.

A computation process, whatever it is, has to be run by a *computer*, which can be a human being using pen and paper, or a *machine* specifically built for that purpose. The most popular form of computer is an electronic device that use a digital representation of data, and manipulate this representation according to a set of *instructions* that implements the algorithm that transforms them into *results*. The set of instructions is then called a *computer programme*.

Electronic computers use numbers, integer and floating point, to represent data. These numbers are commonly coded in base 2, which can also be directly used to encode boolean values and therefore easily implement *conditional* calculations. Electronic computers are mainly built from basic blocks, *logic gates*, that are interconnected to make the arithmetic and logic units, memory registers and micro-controllers that form the *Central Processing Unit* which in turn, along with the *Main Storage Unit*, and the *I/O Controllers* constitute the computer itself.

Therefore, one can build a digital computer using any technology that can mimic the logic gates and their interconnections. We will demonstrate in this article how to implement single logic gates using synthetic minimal biological systems embedded in a vesicle (*protocell*) and how to connect them together to get a device (*protocellular machine*) that computes a complex logical function. The computing model that underlies our biochemical implementation of a computer is similar to the one of an electronic computer, their computing capabilities are the same.

The fundamental characteristic of electronic computers is their ability to run a potentially infinite number of algorithms doing a wide variety of computations on data, because they are *programmable*: the same computer can run sequentially (or *pseudo*-concurrently) as many different programmes as those that can reside in its main memory storage, along with the associated data.

Here, we will show how to build a reduced kind of computer that can only solve one problem, but a problem belonging to a class known to be hard to solve: a NP-complete problem.

The *computational complexity theory* explores the feasibility of computational problems, in terms of computing time (or memory space) needed to solve a problem of a given size. In the *von Neumann* based architectures (standard electronic computers) the number of computing elementary steps (instructions) is often used to approximate the computing time, since each instruction takes approximately the same amount of time to be performed.

There are two main classes of computational problems, those that can be solved by a deterministic machine in a number of steps which can be expressed as a polynomial of the problem size (class P), and those that can be solved in polynomial time, but on a *non-deterministic* machine (class NP). Typically decision problems

where (1) a solution can be verified in polynomial time and (2) there is no other known algorithm except generate and verify all the potential solutions, are NP problems. Solving these problems on a von Neumann computer require an exponential number of steps with respect to the problem size.

A NP problem is said to be *NP-complete* if any other NP problem can be transformed into this problem in polynomial time (Karp 1972). In consequence NP-complete problems are more difficult to solve than any other NP problems because if one NP-complete problem is *quickly* solved (in polynomial time) then all the NP problems will be quickly solved. Of course all these complexity classes collapse if P = NP (which is one of the great open conjectures in computer science).

We have chosen the *3-SAT* problem, a variant of the *boolean satisfiability problem (SAT)*, as an example of NP-complete problem (Cook 1971) a protocellular computer can solve elegantly. This is mainly because the very small size of protocells and their 3D packing allow us to build a machine made of billions of logic gates specifically connected to solve a given 3-SAT problem. Another characteristic of our protocellular machines is that they are *disposable* in the sense that once the computation is done for a given set of input values, the machine is no more usable. But the counterpart is that the energy needed for the computation is very low (Sarpeshkar 2010).

Finally, the biochemical nature of the protocellular machines make them very easy to interface with living organisms. For example, they can be used for medical diagnosis to implement biosensing coupled with medical decision algorithm.

2 Methods

2.1 Protocell Logic Gates Definitions

The bottom-up design of biological systems is made possible by the synthetic biology approach that applies engineering principles to biology in order to design standardised biological parts, devices, systems in a systematic and rational manner. Hierarchical abstraction of biological functions enables the assembly at the system level of new biological systems with user-defined functionalities (Purnick and Weiss 2009; Canton et al. 2008; Endy 2005). The behaviour of synthetic systems is predictable and designs can be automatised in silico before attempting to implement them with biological components (Marchisio and Stelling 2009). In addition, the remarkable capacity of biological building blocks to compute in highly sophisticated ways has led scientists to design and engineer biomolecular computers (Benenson 2012). Thus far, most biocomputing has been investigated from the *top down* perspective, that is, by modifying existing organisms (Khalil and Collins 2010). The strategy we propose here, protoputing, is interested in implementing protocells from the bottom-up perspective to perform computation, where very little attention has been given Rasmussen et al. (2009), Luisi and Stano (2011) and Smaldon et al. (2010).

Starting from an abstract operation that is to be computed, one can rationally and systematically choose biochemical species for the implementation (metabolites,

enzymes, nucleic acids...) (Fig. 1a). Standardised and robust biomolecular components and reactions can be engineered, tested and optimised to implement different types of biological functions or computations (Koeppl 2011): simple boolean operations, memory devices, amplifiers, analog to digital converter, oscillators etc. Figure 1b. In addition, this process can be automatised using CAD tools recently developed for that purpose (Koeppl 2011; Rialle et al. 2010; Chandran et al. 2011).

For example, an AND biochemical logic gate taking reduced metabolites as inputs (NADPH and FADH2) can be implemented using a network of 3 different enzymes and 4 different metabolites connected by 3 biocatalytic reactions, and transferring electrons to NADH as an output. In the same way, we can implement a set of standardised *computation units* that recapitulate all boolean logic gates (see Fig. 3 for examples of implementations of AND, NOT and NOR gates). Electron transfer can also be coupled to various output biological functions to produce human readable signals (Fig. 2) or enable the selection of machines with specific behaviour for further analysis. We propose that specific reduction of species can trigger as an output, either luminescence or fluorescence (Candeias et al. 1998) or the transport of a ligand (or its receptor).

Our approach improves modularity of biomolecular computing systems by the fact that biochemical networks implementing boolean logic are encapsulated within synthetic vesicles, or *protocells*, distinguished by their high degree of organisation and control over biological processes provided by the membrane boundary (Elani et al. 2014). Such architecture of insulated computing units allows us to use many instances of the same type of protocell anywhere in the circuit when the same logic gate is needed. Moreover, this enables the connection of multiple layers of

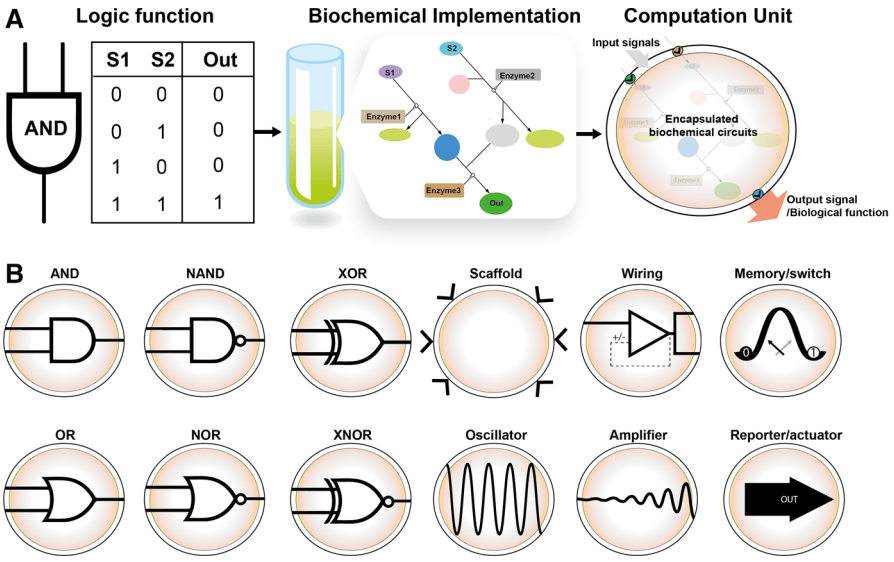

Fig. 1 **a** Rational design of a computation unit implementing a given logical function. (**b** Different types of computation units. An *AND* gate outputs *true* only if the two inputs are *true*; An *OR* gate outputs *true* if at least one of the inputs is *true*; A *XOR* gate outputs *true* only when one of the inputs is *true*; The *NAND, NOR* and *XNOR* gates outputs the opposite value of the *AND, OR* and *XOR* gates respectively

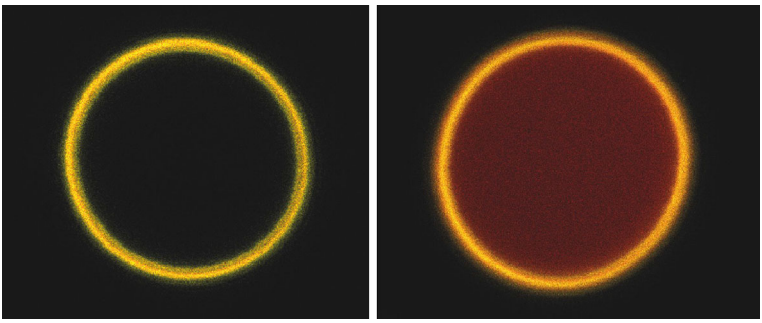

Fig. 2 Example of experimental fluorescence signal triggered in micrometric protocells. Ezymatic electron transfer from carbohydrate to the redox sensor probe (in that case resazurin is reduced into the *red fluorescent* product resorufin). Phospholipidic protocells encapsulating biochemical species were generated using microfluidic devices, and imaged using a confocal microscope. *Left* no induction; *right* induced with glucose. (Color figure online)

protocells to achieve complex information processing capabilities. In such architecture, input information arrives from upstream connections with previous protocells, to output connections to following computation units.

As each logic gate is encapsulated in an impermeable vesicle, the reactions that compute the output value will go from the non-equilibrium initial state to an equilibrium state. Therefore, once a logic gate has finished to compute the output, it is no more able to do another computation. So this first model of protocellular machine is in essence a kind of disposable computer.

Encapsulation of biochemical networks can be achieved using natural bilayer membranes (e.g. phospholipid bilayers, liposomes) (Noireaux and Libchaber 2004), or engineered membranes (e.g. copolymers, polymersomes) (Kamat et al. 2011), with respect to stoichiometry of internal species and incorporation of membrane proteins for connections (Chaize et al. 2004; Huang et al. 2014; Peters et al. 2014). This process is also known to stabilise enzymes, prevent cross-talk, denaturation or proteolysis and improve enzymatic properties (Yoshimoto 2011; Sunami et al. 2010). In addition, streamlined workflows, for example relying on microfluidics, are already available for the high-throughput generation of protocells that encapsulate various substrates (Richmond et al. 2011; Thiele et al. 2010; Duncanson et al. 2012; Matosevic and Paegel 2011; Teh et al. 2011). This strategy, extensively used in our lab, allowed us to test the implementation of various protocellular logic gates. Such vesicle have proven to be sufficently stable (i.e. not prone to fuse together or physical disruption) to enable the construction of such multi vesicular assemblies (Stanish and Singh 2001; Teh et al. 2011). Tunable sizes ranging from 50 nm to 50 µm can be obtained, although in our approach, size should be kept as small as possible to obtain the highest density of computing operators.

2.2 Circuit Wiring

To obtain a full circuit implementing a given boolean function, we then need to concatenate and wire basic logic gates. The design of a function-specific

protocellular machine exploits the composability of *computation units*. Amongst a specific set of protocells, multiple instances of the same logic gates can be *wired* together to implement a user-defined function.

One way to achieve successive reactions in each layer of a protocellular machine, from input to output protocells, is to drive them using electrochemical potential (e.g. oxido-reduction reactions). By analogy with electronic computers, electrons are energy carriers and the redox potential is the *current* of the system, which could be measured with an electronic device. The major difference is that inside a protocell, wires are replaced by free molecules (e.g. NADH, NADPH, FADH2), and effective wiring is achieved using chemical selectivity of enzymes. Molecules are either electron donors or acceptors, obeying biological enzymatic rules resulting in current and energy for computation. In such systems, the *in* → *out* direction is driven by the thermodynamics of the redox reaction. In our example, a protocell giving the *true*value would have a reductive state with high concentration of NADH, which can then transfer its electron to reduce the input of the next protocell. Conversely, a protocell giving the *false*value does not output any electron. In addition, electron transfer occurs only between physically connected protocells, through tight junctions putting into close contact electron transfer complexes, which carry out the connections between protocells and therefore between logic gates (Fig. 3).

We will build a protocellular machine from a set of protocell logic gates assembled in a tree-like layout (see Sect. 3). When set to *true*, the inputs of the machine initiate electron transfer through the chain of protocells that constitutes each branch of the tree, down to the root protocell.

In these input protocells, electron production is started by the specific oxidation of molecular species by oxidase enzymes. Electrons are then transferred down the protocell chain via transmembrane electron transport complexes that enable electron coupling (reduction) of specific molecular species. In that sense, input protocells can be seen as the *generators* that power the machine. Moreover *fuel* protocells, with a switch like behaviour, could be used to amplify and reshape the signal and therefore counteract its decay.

In order to implement specific electron transfer modules, we propose to exploit the modularity and thermodynamic reversibility of natural oxidative phosphorylation and photosynthesis complexes, which catalyse the electron transfer across natural membranes with specificity to NADH (Complex I), FADH2 (complex II), and NADPH (NADPH quinine oxido reductase) (Osyczka et al. 2004). This includes quinone (or chemically related) and cytochrome c shuttle, which are delocalised mobile electron carriers that could be used as inter-protocell transfer molecules. In our design, we propose that a first quinone carrier (or related), could transfer electrons from a specific output signal (substrate specificity given by the first complex: I, II...) to a close complex III, which would then via a mobile cytochrome c transfer these electrons forward to the complex III belonging to the next protocell. This mechanism constitutes efficient reversible energy coupling, which has been shown to work via electron-tunneling across the proteins (Osyczka et al. 2004). Furthermore, recent studies have highlighted the possibility to re-engineer natural prokaryotic complexes for efficient and substrate specific synthetic

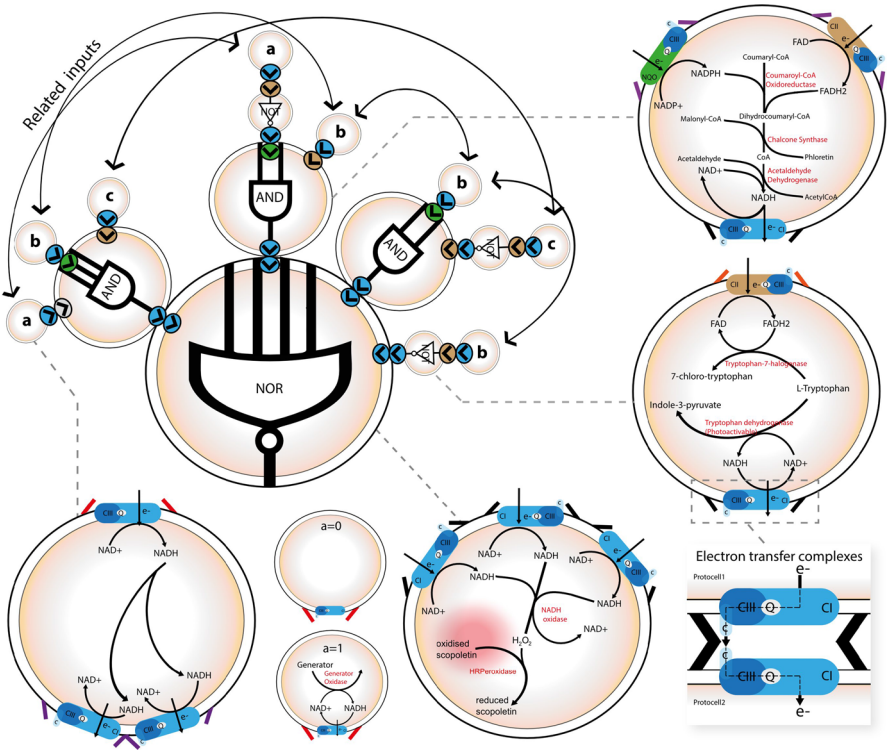

Fig. 3 Detail of a possible implementation of each type of protocell gate. Each type of logic gates has been simulated in silico with HSIM, and some of them are under test in the lab). The detail of the electron transfer mechanism is shown in the bottom right cartoon. For example, the fluorescent NOR gate uses a cascade of two enzymatic reactions (NADH oxidase, Horseradish Peroxidase) to consume the fluorescent oxidised scopoletin when NADH is present in the protocell, that is when at least one input is set to *true*, so is transferring electrons to make NADH from the initial pool of NAD^+

electron transporters (Katzen et al. 2002; Page et al. 1999; Wakeham and Jones 2005).

The architecture of a machine is controlled by the functional *wiring* of input and output of specific protocells. This can be achieved by using programmable junction modules, that can be selected to implement any protocellular machine in a *plug-and-play* way (Fig. 3). Biological function for these programmable attachments could be supported by couples of ligand/receptors with high binding affinity, such as aptameric nucleic acids (Hermann and Patel 2000; Smuc et al. 2013) or peptidic binders (Falciani et al. 2005), that could be straightforwardly produced in large combinatorial synthetic libraries using SELEX (Stoltenburg et al. 2007), or ribosome display respectively (Hanes et al. 2000; Binz et al. 2005).

Starting from a pre-built stock of computation units, the user can define a set of attachment instructions that corresponds to the boolean function to implement. Irreversible constructs can be achieved using cross-linking chemicals, so that no unbinding would occur (Song et al. 2012; Xiang et al. 2014). We assume that the

kinetics associated with such an assembly process would be of the order of minutes. Some attachments can also be set as *random*, to enable stochastic wiring of different types of protocells to specific positions. This could be used for example to solve problems involving the navigation through a large parameter space where protocellular machines could be used to compute a fitness function. Additonnally selection methods could be implemented to isolate protocellular machines that exhibit specific behaviours. Positive selection can be done for example using FACS, conversely negative selection via a self-destruction mechanism.

3 The Case Study

3.1 Boolean Satisfiability Problem

The NP-complete problem we aim to solve is the *3-SAT* problem. This problem can be simply defined as:

> given any boolean formula in Conjonctive Normal Form (CNF), with at most 3 litterals per clause, is there a valuation of the variables that satisfy the formula?

In other words, it asks whether the variables of a given boolean formula can be consistently replaced by the values *true*or *false*in such a way that the formula evaluates to *true*. If it is the case, the formula is called satisfiable. The litterals are either a variable (v) or the negation of a variable ($\neg v$); They are connected with the *or* operator ($\vee$) to form a clause; The clauses are connected with the *and* operator ($\wedge$) to obtain the formula in CNF. For example:

$$F(a,b,c) = (a \vee \neg b \vee c) \wedge (b \vee \neg c) \wedge (a \vee b) \tag{1}$$

is true when $a = true, b = true$ and $c = false$, so the formula $F(a,b,c)$ is satisfiable. Conversely, the formula:

$$G(a,b,c) = (a \vee b \vee c) \wedge (\neg a \vee b) \wedge (b \vee \neg c) \wedge \neg b \tag{2}$$

is not satisfiable because all the eight possible valuations for a, b, c lead to $G = false$.

To find if a formula is satisfiable, we will build as many protocellular machines as there are combinations of valuations of the input variables. To do this, we will exploit the combinatorial power of ligand-receptor binding to link constant protocells (with *false*or *true*values) to the inputs of the protocellular machine to cover all the value space. A protocellular machine is dedicated to a specific formula, and therefore is not programmable in the sense an electronic computer is. The protocellular machines are self assembled according to the formula they have to check, so in our approach, the *programme* is the process that directs the assembly of the machines. We will ascertain that there is at least one instance of a protocellular machine per possible valuation of the variables.

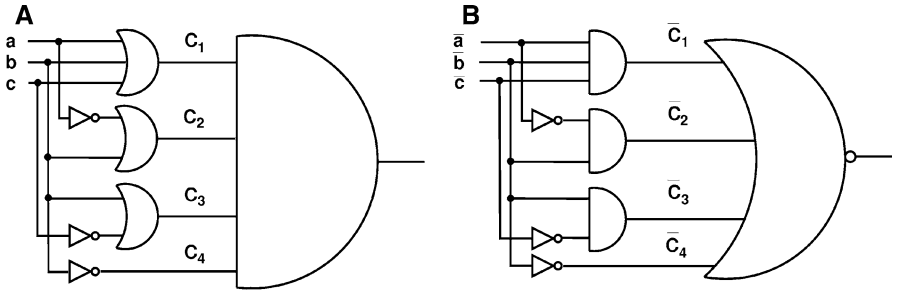

Fig. 4 **a** Direct implementation of the G formula in standard Conjunctive Normal Form. **b** Using the De Morgan laws, the same boolean function is rewritten using a NOR gate instead of the final AND gate, easier to build with a large number of inputs, and multiple 2- and 3-inputs AND gates fed with the complement of the original inputs

An instance of the machine can be made using 2- and 3-inputs OR gates connected to a big AND gate with as much inputs as there are clauses in the formula. Each input of a clause is connected to a protocell representing a variable *v* sending *true*or *false*when a specific start signal is given, or to an inverter protocell sending the negation of *v* when the start signal is given. The output of the AND gate is connected to a protocell that fluoresces when the input value is *true*. For example, the protocellular machine corresponding to the *G* formula would be made of a 4-input AND gate, two 2-input OR gates, one 3-input OR gate and three inverters connected as in the equation above (Fig. 4a).

As we have at least one (and probably more) instance of the machine for each possible valuation of the variables, if at least one of the protocellular machine fluoresces, the formula is satisfiable. Conversely, if there is no fluorescence at all then the formula is not satisfiable.

We can simplify the construction of the machines using the De Morgan laws to replace the big AND gate by a NOR gate, which is easier to build and also more efficient than an AND gate when there is a lot of inputs. Since the output of this NOR gate is the output of the whole machine, the final inverter can be made using an inhibitor of the fluorophores stored inside the protocell implementing the gate. We also need to feed the inputs of the AND gates with the complement of the variables, which could lead us to use a lot of inverters; But they can be avoided because these inputs are the inputs of the whole machine, and since we need to test all the valuations of the variables, these inputs will be fed with *constant* values. Therefore we can program the assembly of a machine with the constants already inverted (Fig. 4b) and we will need no more inverters than negated variables specified in the original formula.

3.2 The Assembly of the Machines

To obtain one instance of a computing protocellular machine, we need to direct the self assembly of as many copies of AND gate protocells as there are clauses in the formula (except when a clause has only one litteral), the output of each AND gate

being connected to an input of a fluorescent NOR protocell. The inputs of each AND gate are also to be connected to the output of an inverter or to the output of a wiring protocell (representing the input variables of the formula). Then, to test a valuation of the variables of the formula, the input of each wiring protocell will be connected to special inputless protocells that output the constant value *true*or *false*. Once the machine and its inputs are assembled, when a *start signal* is given, after a few minutes, the NOR gate of this machine will fluoresce if the formula is *true*for this valuation of the variables, and therefore the formula is satisfiable.

We must ensure that *correlated* inputs of two (ore more) AND gates are fed with correlated values. In the previous formula (rewrited using a NOR of ANDs, with the complemented variables as shown in Fig. 4b)

$$G(a,b,c) = \overline{(\overline{a} \wedge \overline{b} \wedge \overline{c}) \vee (\neg\overline{a} \wedge \overline{b}) \vee (\overline{b} \wedge \neg\overline{c}) \vee \neg\overline{b}} \tag{3}$$

the first input of the first clause, $\overline{a}$, is always the opposite of the first input of the second clause ($\neg\overline{a}$), and the second input of the two first clauses, $\overline{b}$, have always the same value, etc. To achieve that we will use *inverter* protocells, and *wiring* protocells that can transfer their input to two or more outputs.

In this example, since there are 3 variables, we must assemble 8 protocellular machines to test each of the 8 possible valuations. Each line of the table in Table 1 shows the input values (0 for *false*, 1 for *true*) of one of the 8 different protocellular machines, the complemented value of each clause, and the value of the formula (3), which is always *false* (this formula is not satisfiable).

In order to have a efficient assembly mechanism, we split the process in two steps. The first one does not depend on a specific formula, but on the maximal numbers of variables ($V_{\max}$) and of clauses ($C_{\max}$) a formula can have. To be able to test any given formula within the limits of size we stated, we build a reservoir containing at most for one protocellular machine instance:

- one $C_{\max}$-input NOR gate
- $C_{\max}$ 2- and 3-inputs AND gates.
- $V_{\max}$ *inverter* protocells
- $2 \cdot V_{\max}$ types of inputless *constant protocells*, outputting the constant *false*or *true*to represent the two possible values of each variable.

Table 1 Complemented value of each clause for the eight possible valuations of the variables, and the corresponding value of the formula

$\overline{a}$	$\overline{b}$	$\overline{c}$	$\overline{c}_1$	$\overline{c}_2$	$\overline{c}_3$	$\overline{c}_4$	$G(a,b,c)$
0	0	0	0	0	0	1	0
0	0	1	0	0	0	1	0
0	1	0	0	1	1	0	0
0	1	1	0	1	0	0	0
1	0	0	0	0	0	1	0
1	0	1	0	0	0	1	0
1	1	0	0	0	1	0	0
1	1	1	1	0	0	0	0

- a formula dependent number of *wiring* protocells that duplicate their input to two (or more) outputs in order to cast each constant protocell output to the appropriate AND input or inverter.

Of course we can have a larger number of copies of these building blocks if we want to test more than one instance of the formula.

We can remark that depending on the formula we want to test, all the $C_{\max}$ inputs of the NOR gate are not used and will stay not connected to any output, which is equivalent to a *false*value and so these inputs will not interfere with the computation since we are certain that nothing can be bound to them.

To verify the satisfiability of a formula made of $N < V_{\max}$ variables and $C < C_{\max}$ clauses, we need to build 2^N protocellular machine instances, (at least) one per possible valuation of the input variables. The building of these protocellular machines constitutes the second step. Although this step is specific to a given formula, its principle is generic enough to be applied to any formula. This resemble to the *compilation* phase of a programme written in a high level programming language on a standard computer.

To assemble a machine we will *program* the binding of each input of one NOR gate to the output of a 2-inputs or a 3-inputs AND gate, or to one output of a wiring protocell, or to the output of an inverter. We will also need to program the binding of one wiring protocell per variable to some inverter, AND or NOR input, according to the formula. Then, to test a given valuation of the variables, we will need to bind the constant protocells corresponding to each variable of the formula to the inputs of this machine.

These programmed bindings are made possible because all the protocells in the reservoir have been built with specific tags on their inputs and outputs. These tags can be peptides/nucleic acids with a unique sequence to address them. The process of binding itself will be done by putting in the environment specific molecular attachment instructions that recognise and bind the tag on the output and the tag on the corresponding input. This will enable the binding of specific protocells together (Fig. 5).

Each input of the NOR gate is labeled with a tag implementing the number of the corresponding clause (0 to $C_{\max} - 1$). Similarly the output of each of the AND gate is labeled with the same number. Therefore, to connect an AND gate to the corresponding input of the NOR gate for one protocellular machine, we have to synthesise an molecular attachment that match at one end the tag labelling the output of the AND gate and at the other end, the tag labelling the input of the NOR gate.

The same mechanism is used for the input variables of the formula. The input of a wiring protocell that corresponds to a variable of the formula is labeled with a tag representing the variable number (0 to $V_{\max} - 1$). The constant protocells used for each variable, whether their output is *false*or *true*, are labeled with a tag matching the corresponding wiring protocell of the machine. Since there is a high number of constant protocells in the medium, the *false*and *true*version for each variable will be randomly bound to the corresponding input of the machines, and after some time, all the possible valuations will be covered.

It is important to notice that we must use constant protocells that output the boolean value *false*, even if a non-connected input is equivalent, because when we

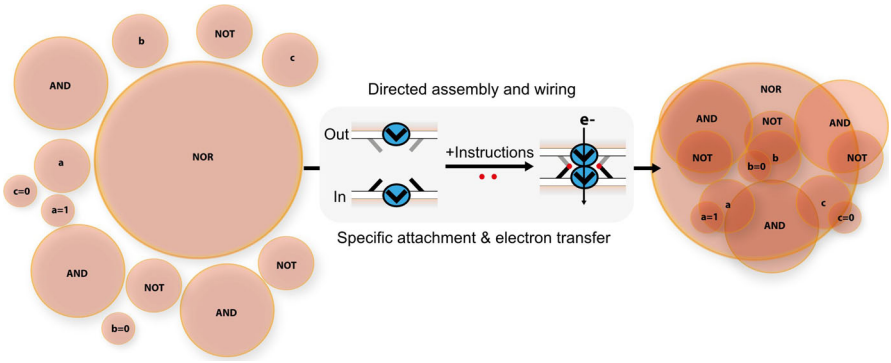

Fig. 5 Directed assembly and wiring via specific attachments of one instance of a protocellular machine for the formula G (a, b, c). The inputs a, b and c are implemented with *wiring* protocells (one input, one, two or more outputs) that distributes the values of the variables to inverters or to the NAND gates according to the formula (3), see Fig. 4b. The NOR gate is a large protocell underneath the AND gates, where the outputs of the AND gates are bound. The small protocells $a, b, c = 0, 1$ are constant protocells for the input variables a, b and c (*left*). These input protocells will be randomly be bound to constant *false*or *true* protocells to cover all the valuations of the variables. On the *right side*, the protocellular machine assembled tests the valuation $a = 1, b = 0, c = 0$

want to test a valuation where some variable is *false*, we must be certain that no *true*constant protocells can be bound to this input.

3.3 The Computation Process

The computation process may begin when we are certain that at least one copy of a protocellular machine is bound to each possible combination of input values.

This process is started by remotely triggering the whole population of *true*constant protocells and inverter protocells using, for example, light switchable enzymes (a tryptophan dehydrogenase engineered to bear a photoswitch moiety) (Strickland et al. 2012; Rakhit et al. 2014; Riggsbee and Deiters 2010).

Since all the machines run concurrently to compute the value of the formula, the total computing time is the time needed either by the first one that output *true* (that become fluorescent) or when we can be certain that the slowest machine that outputs *false*has finished (in this case they all do). If there is a small number of protocellular machines that fluoresces, we could enhance the signal/noise ratio by scattering the solution into several parts such that the concentration of the fluorescent machines would appear higher, and so helps its detection. Another way to easily detect the first (and possibly only) protocellular machine that outputs *true*would be that this machine triggers the fluorescence of those in its neighbourhood, and so increase the global fluorescence. Independently of the formula we want to test, the maximal number of reactions needed from one input to the output is very small: one inverter, a small number of wiring protocells, one AND, and one NOR.

Considering the kinetics of enzymatic processes for these simple reactions, we could assume that the calculation time of a single protocell (i.e. the time required for effective electron transfers though the protocell) would be in the order of a few minutes. The computation time for one protocellular machine would then be proportional to the number of layers of this machine. The total computing time would not exceed 20 min, whatever the number of protocellular machines is needed to solve the problem. This is of course mainly because the computing process is massively parallel and to a lesser extent because each *processor* is dedicated to the specific problem we want to solve.

Since the size of a complete protocellular machine is of the order of magnitude of a micron-cube, even less, we can have more than 10^{12} machines in a few ml of solution. As 10^3 is approximately equal to 2^{10}, we could theoretically have about $2^{10(12/3)} = 2^{40}$ machines in a few ml. Therefore using this technique, we could potentially solve any 3-SAT problem involving up to 40 variables in a few minutes. If we suppose that an electronic computer needs 1 μs to generate and test one valuation of the variables, the average computing time would be of the order of $10^{12} \cdot 10^{-6} = 10^6$ s, which is more than 11 days and a half.

Moreover, if we suppose we use a low power electronic computer, for example 20 watts, the energy consumed at the end of the 11.5 days would be $10^6 \cdot 20 = 2 \cdot 10^7$ J (≈5.5 kWh), compared to a few joules for the protocellular machines.

4 Conclusion

The case studied here is an example of what we could do with protocellular machines, and how to make them. Of course, making the huge number of instances of protocellular machines needed to verify the satisfiability of a large formula is a bit speculative at the present day, but the mechanisms used to engineer their building blocks and to direct their assembly are already under test in the lab. Many implementations of logic gates (much more than those shown in Fig. 5) have been tested in silico using the HSIM (Amar et al. 2008) simulation system and proven to be functionning (Bouffard et al. 2015).

The computing time we claim, approximately one thousand times faster than a traditional electronic computer for a specific class and size of problem, is also a bit provocative, but the fact remains that this is an example of how to use the really massive parallelism of protocellular machines in order to solve dedicated problems. Moreover, to our knowledge, this is the first case where a synthetic biochemical computer could realistically compete with the speed of electronic computers, while being far less demanding in terms of energy.

Nevertheless, in our opinion, the most exciting perspective of protocellular machines is that they are electronically and biologically interfaceable. Thus they could be incorporated in living organisms, or into hybrid electronic/biological systems. Our approach allows us to design any given boolean function that can be

connected and triggered by any biological and/or electronical input, and generate chosen outputs in a similar way.

References

Amar P, Legent G, Thellier M, Ripoll C, Bernot G, Nystrom T, Saier M Jr, Norris V (2008) A stochastic automaton shows how enzyme assemblies may contribute to metabolic efficiency. BMC Syst Biol 2:27

Benenson Y (2012) Biomolecular computing systems: principles, progress and potential. Nat Rev Genet 13(7):455–468

Binz HK, Amstutz P, Plückthun A (2005) Engineering novel binding proteins from nonimmunoglobulin domains. Nat Biotechnol 23(10):1257–1268

Bouffard M, Molina F, Amar P (2015) Extracting logic gates from a metabolic network. In: P. Amar, F. Képès, V. Norris (eds.) Proceedings of the conference "advances in systems and synthetic biology". EDP Sciences, Strasbourg, France, pp 63–76

Candeias LP, MacFarlane DPS, McWhinnie SLW, Maidwell NL, Roeschlaub CA, Sammes PG, Rachel W (1998) The catalysed nadh reduction of resazurin to resorufin. J Chem Soc Perkin Trans 2:2333–2334

Canton B, Labno A, Endy D (2008) Refinement and standardization of synthetic biological parts and devices. Nat Biotechnol 26(7):787–793

Chaize B, Colletier J-P, Winterhalter M, Fournier D (2004) Encapsulation of enzymes in liposomes: high encapsulation efficiency and control of substrate permeability. Artif Cells Blood Substit Immobil Biotechnol 32(1):67–75

Chandran D, Bergmann FT, Sauro HM, Densmore D (2011) Computer-aided design for synthetic biology. In: Koeppl H, Setti G, di Bernardo M, Densmore D (eds) Design and analysis of biomolecular circuits. Springer, New York, pp 203–224

Cook SA (1971) The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on theory of computing, STOC'71, New York, NY, USA, ACM, pp 151–158

Duncanson WJ, Lin T, Abate AR, Seiffert S, Shah RK, Weitz DA (2012) Microfluidic synthesis of advanced microparticles for encapsulation and controlled release. Lab Chip 12(12):2135

Elani Y, Law RV, Ces O (2014) Vesicle-based artificial cells as chemical microreactors with spatially segregated reaction pathways. Nat Commun 5:5305

Endy D (2005) Foundations for engineering biology. Nature 438(7067):449–453

Falciani C, Lozzi L, Pini A, Bracci L (2005) Bioactive peptides from libraries. Chem Biol 12(4):417–426

Hanes J, Schaffitzel C, Knappik A, Plückthun A (2000) Picomolar affinity antibodies from a fully synthetic naive library selected and evolved by ribosome display. Nat Biotechnol 18(12):1287–1292

Hermann T, Patel DJ (2000) Adaptive recognition by nucleic acid aptamers. Science 287(5454):820–825

Huang X, Patil AJ, Li M, Mann S (2014) Design and construction of higher-order structure and function in proteinosome-based protocells. J Am Chem Soc 136(25):9225–9234

Kamat NP, Katz JS, Hammer DA (2011) Engineering polymersome protocells. J Phys Chem Lett 2(13):1612–1623

Karp RM (1972) Reducibility among combinatorial problems. In: Miller RE, Thatcher JW (eds) Complexity of computer computations. Plenum Press, New York, pp 85–103

Katzen F, Deshmukh M, Daldal F, Beckwith J (2002) Evolutionary domain fusion expanded the substrate specificity of the transmembrane electron transporter DsbD. EMBO J 21(15):3960–3969

Khalil AS, Collins JJ (2010) Synthetic biology: applications come of age. Nat Rev Genet 11(5):367–379

Koeppl H (2011) Design and analysis of bio-molecular circuits. Springer, Berlin

Luisi PL, Stano P (2011) The minimal cell the biophysics of cell compartment and the origin of cell functionality. Springer, Berlin

Marchisio MA, Stelling J (2009) Computational design tools for synthetic biology. Curr Opin Biotechnol 20(4):479–485

Matosevic S, Paegel BM (2011) Stepwise synthesis of giant unilamellar vesicles on a microfluidic assembly line. J Am Chem Soc 133(9):2798–2800

Noireaux V, Libchaber A (2004) A vesicle bioreactor as a step toward an artificial cell assembly. Proc Nat Acad Sci USA 101(51):17669–17674
Osyczka A, Moser CC, Daldal F, Leslie Dutton P (2004) Reversible redox energy coupling in electron transfer chains. Nature 427(6975):607–612
Page CC, Moser CC, Chen X, Dutton PL (1999) Natural engineering principles of electron tunnelling in biological oxidation-reduction. Nature 402(6757):47–52
Peters RJRW, Marguet M, Marais S, Fraaije MW, van Hest JCM, Lecommandoux S (2014) Cascade reactions in multicompartmentalized polymersomes. Angew Chem Int Ed 53(1):146–150
Purnick PEM, Weiss R (2009) The second wave of synthetic biology: from modules to systems. Nat Rev Mol Cell Biol 10(6):410–422
Rakhit R, Navarro R, Wandless TJ (2014) Chemical biology strategies for posttranslational control of protein function. Chem Biol 21(9):1238–1252
Rasmussen S, Bedau MA, Chen L, Deamer D, Krakauer DC, Packard NH, Stadler PF (2009) Protocells: bridging nonliving and living matter. MIT Press, Boston
Rialle S, Felicori L, Dias-Lopes C, Peres S, Atia SE, Thierry AR, Amar P, Molina F (2010) BioNetCAD: design, simulation and experimental validation of synthetic biochemical networks. Bioinformatics 26(18):2298–2304
Richmond DL, Schmid EM, Martens S, Stachowiak JC, Liska N, Fletcher DA (2011) Forming giant vesicles with controlled membrane composition, asymmetry, and contents. Proc Nat Acad Sci 108(23):9431–9436
Riggsbee CW, Deiters A (2010) Recent advances in the photochemical control of protein function. Trends Biotechnol 28(9):468–475
Sarpeshkar R (2010) Ultra low power bioelectronics: fundamentals, biomedical applications, and bio-inspired systems. Cambridge University Press, Cambridge
Smaldon J, Romero-Campero FJ, Trillo FF, Gheorghe M, Alexander C, Krasnogor N (2010) A computational study of liposome logic: towards cellular computing from the bottom up. Syst Synth Biol 4(3):157–179
Smuc T, Ahn I-Y, Ulrich H (2013) Nucleic acid aptamers as high affinity ligands in biotechnology and biosensorics. J Pharm Biomed Anal 81–82:210–217
Song S, Kole S, Bernier M (2012) A chemical cross-linking method for the analysis of binding partners of heat shock protein-90 in intact cells. BioTechniques. doi:10.2144/000113856
Stanish I, Singh A (2001) Highly stable vesicles composed of a new chain-terminus acetylenic photopolymeric phospholipid. Chem Phys Lipids 112(2):99–108
Stoltenburg R, Reinemann C, Strehlitz B (2007) SELEX-a (r)evolutionary method to generate high-affinity nucleic acid ligands. Biomol Eng 24(4):381–403
Strickland D, Lin Y, Wagner E, Hope CM, Zayner J, Antoniou C, Sosnick TR, Weiss EL, Glotzer M (2012) TULIPs: tunable, light-controlled interacting protein tags for cell biology. Nat Methods 9(4):379–384
Sunami T, Hosoda K, Suzuki H, Matsuura T, Yomo T (2010) Cellular compartment model for exploring the effect of the lipidic membrane on the kinetics of encapsulated biochemical reactions. Langmuir 26(11):8544–8551
Teh S-Y, Khnouf R, Fan H, Lee AP (2011) Stable, biocompatible lipid vesicle generation by solvent extraction-based droplet microfluidics. Biomicrofluidics 5(4):044113
Thiele J, Abate AR, Shum HC, Bachtler S, Förster S, Weitz DA (2010) Fabrication of polymersomes using double-emulsion templates in glass-coated stamped microfluidic devices. Small 6(16):1723–1727
Wakeham MC, Jones MR (2005) Rewiring photosynthesis: engineering wrong-way electron transfer in the purple bacterial reaction centre. Biochem Soc Trans 33:851–857
Xiang Z, Lacey VK, Ren H, Jing X, Burban DJ, Jennings PA, Wang L (2014) Proximity-enabled protein crosslinking through genetically encoding haloalkane unnatural amino acids. Angew Chem Int Ed 53(8):2190–2193
Yoshimoto M (2011) Stabilization of enzymes through encapsulation in liposomes. In: Minteer SD (ed) Enzyme stabilization and immobilization. Humana Press, New York, pp 9–18