Embodied reaction logic in a simulated chemical computer

Fintan Nagle[1] and Simon Hickinbotham[2]

[1]UCL CoMPLEX, University College London, London WC1E 6BT
[2]YCCSA, Department of Computer Science, University of York, Heslington, York YO10 5GH, UK
fintan.nagle.10@ucl.ac.uk sjh@cs.york.ac.uk

Abstract

This work uses an ALife simulation to explore the implementation of embodied reaction logic in a chemical computer. Chemical systems have potential for computation. There are properties of a logical system that are desirable in any computational system, such as the ability of the system to change state in response to some input. An issue in chemistry is that the molecules must have some physical embodiment, which must somehow represent state; state is then interpreted as the presence or absence of certain molecular configurations in the system. The design of a chemical logic gate is a means of showing that a chemical system can change state appropriately and that the information encoded in the molecules is available to be processed *as* information. This paper compares two simulated chemical computing systems: Bindworld (a simple illustrative example) and Stringmol, (a fully implemented complex DNA-inspired evolutionary computing framework). The problems and design decisions involved in creating a NOT gate in each system are compared, showing that designed computational systems require a certain complexity and flexibility to be useful to human operators. Finally we discuss general extensions to the Stringmol reaction chemistry that would simplify the process of information processing in embodied systems.

Computation is a fairly new concept to science. Although the word itself has been in use since 1447 or before, until the early 20th century it referred only to manual calculation performed by humans (this is why early machines were called "automatic computers" to distinguish them from their human counterparts). It is only since the development of the mechanical and electronic computer that the term has been applied to a process external to human thought.

Artificial Life (ALife) is a simulation of biological life on a computer. These simulations are often considered to be "embodied thought experiments" (Di Paolo et al., 2000), which test whether the essential properties of biology have been captured. Simulations of biological processes are also seen as a step towards harnessing biological processes for our own ends (Brooks, 2001). (The successful simulation will process information in a similar, but more robust, manner to our electronic computers.) It is therefore legitimate to consider how a biological simulation is capable of information processing. The simplest form of this is logic.

Models of conventional computing deliver programming languages, based on logic, that abstract the functionality away from the implementation of the logic on electrical circuitry. Programmers take this for granted when designing software using these programming languages. Similar abstractions may be needed to program computers based on other media. Generally, computation proceeds in the following context:

1. **`INPUT`**: An observer encodes a problem and passes it to some external system (be it electronic, neural, biological, molecular, or otherwise) via a *setup function*.
2. **`COMPUTE`**: The system evolves, ending in some changed state. Where the change in state has involved some notion of information processing, *computation* has occurred.
3. **`OUTPUT`**: The observer uses an *output function* to extract some useful information from the system and decode it into a useful response.

The problem, or series of instructions, is encoded in a different way depending on the architecture of the computer. In electronic computers it is a program in a language such as C (or, equivalently, the machine code representation of that program). The usefulness of a computer is in the **`COMPUTE`** stage, when the computer performs a task so that we do not need to execute it ourselves.

Systems requiring two-way data exchange (such as a search engine, which alternates between taking queries and returning results) can be seen as a series of input-compute-output operations. We do not discuss parallelism and concurrency, but restrict our discussion to an input-compute-output problem analogous to a batch processing command.

The idea of a chemical computer is appealing: somehow encode the task in a solution of *input molecules*, place them in solution with the *computational molecules*, and find the result in the set of extractable *output molecules*. Molecular computing is massively parallel - there is potential for billions of reactions to take place at the same time in a single container. For example, in DNA computing (Păun et al., 1998; Lee et al., 2004; Adleman, 1994), the setup function consists of encoding a problem in fragments of DNA. The

computation occurs *in vitro*, in parallel, and without external interference. The output function often consists of using gel electrophoresis to extract and sequence DNA of a particular weight, which we know to encode a useful response. Importantly, the information is *embodied* – it is represented as a sequence of DNA that is processed via the laws of physical chemistry.

This paper compares two embodied computers. The first is a small theoretical novel computer, Bindworld, in which simple "atoms" bind with each other to form complexes. The second is a larger, implemented novel computer, Stringmol, in which long RNA-like sequences interact according to their embedded programs. We use the comparison to highlight the challenges and tradeoffs involved in designing a novel embodied computer.

It is important to emphasise the role of the molecules-as-programs in terms of their computing power. To demonstrate the importance of reactions within a chemical computation, we briefly describe some theoretical work based on the concept of complementary binding of molecules alone. We show the shortcomings of this approach, and extend it to incorporate the concept of reaction between molecules once they have bound together before presenting our experimental implementation of a NOT gate in an artificial chemistry.

Stringmol is an *artificial chemistry* (AC) that is being developed to explore a method of evolutionary computation based on the "RNA-world" model of biology (Gilbert, 1986). This is a computational simulation in which the genome-carrier molecule is composed of the same molecular building blocks as the enzymes that are encoded therein. The system, called Stringmol (Hickinbotham et al., 2010a,b, 2009) abstracts the concept of stochastic mixing and molecular binding and reaction into a tractable model for ALife experiments. The link between computation and open-ended evolution is that both paradigms require that it is possible to generate an unbounded set of possible states. In related work (Clark et al., 2011), we have demonstrated that the sophisticated binding protocols in Stringmol are key to the diversity that the system is capable of producing. Here, we show that binding *alone* is not a convenient form for computation.

In addition to being able to carry out computation, it must also be feasible for a human programmer to initialise the system "by hand". The idea of logic gates is an appropriate area in which to start thinking about a novel computational system, since logic is familiar and universally used in traditional computational systems. A non-standard computer which can be used with a standard computing paradigm (logic) is more approachable than one which requires an entirely new way of thinking about computation; it requires less training to use and program, and existing results and algorithms are easier to re-use. Conceptualising such a familiar idea in a new system highlights the similarities and differences between the novel system and von Neumann computing. The two systems we compare here are both formally specified (Bindworld by its reaction rules, Stringmol by its program code); these mathematical and programmatical specifications are not included here, however, as conceptual analysis of the two systems informs our main conclusions.

Logic gates have already been implemented in a real chemical system based on DNA enzymes with catalytic action (Stojanovic and Stefanovic, 2003). Simulation is an important complement to experiments with real chemistry, since the simulation can be interrogated easily and completely, and complete understanding of the molecular model is available. The issue with simulation is whether the correct properties of real chemistry have been captured in the model. The systems we compare here differ from real chemistry: Bindworld is much more simple (containing only atoms and bonds); and Stringmol uses a programming metaphor in the place of the physical properties of atoms by containing a set of operators and pointers.

Teuscher provides a good review of realisations of logic elements in chemical computers (Teuscher, 2007), focusing on how to build in reliability through redundancy in membrane-based systems. The systems we explore here, however, are different from membrane computing systems; they lack a container-based physical hierarchy.

Two important properties of a useful computational system are *preservation of state* and *change of state*. Firstly, information must be preserved in some way; the system must have some kind of memory. Secondly, information must be modified in some sensible way; a system that does not change cannot perform computation. Logic encapsulates both the idea of preservation of state (truth values are held steady) and that of change of state (output values are modified).

What are we looking for in a non-standard computational system?

We already possess a remarkably powerful and ubiquitous computational system: the von Neumann architecture (Aspray, 1990) implemented on electronic computers. This architecture is used by virtually all electronic computing devices, from the mobile phone to the supercomputer. There is therefore little point in developing and researching a novel computational system unless it is (or has the potential to be) in some way superior to the von Neumann electronic computer. Simulations are useful for research, but often difficult to implement on traditional computers. A platform amenable to evolution is also desirable. Thus, we require a different form of embodiment to that seen in electronic computers. We seek a general computational platform that is more amenable to biological systems in general, and biological evolution in particular.

The role of a computer is to carry out algorithms for humans, or even adjust these algorithms to cope with particular problems; informally, *computers solve problems* for us.

The Church-Turing thesis (which is widely regarded and accepted as correct, but is unproved, and indeed formally unprovable (Copeland, 1996)) states that every expressible or comprehensible algorithm is computable using a Turing machine. As electronic computers are equivalent to Turing machines, no novel computer will ever allow *more* problems to be solved than an electronic computer.

One advantage of a novel computer could be that there were some problems it could solve better (either faster or more accurately). Another could be that the **INPUT** stage were easier; that the problem encoding were more understandable or easier to generate. This is very important, as much of the effort involved in computer-aided problem solving goes into formalising the problem in a way that the computer can understand. Finally, a novel computer might not allow problems to be solved faster or more easily, but might offer practical advantages like being smaller, lighter, cheaper, or more energy-efficient. A novel computer might even be able to solve fewer problems than an electronic computer, but convey significant practical advantages.

In vitro experimentation versus *in silico* simulation

We can explore non-standard computational systems in two ways: by implementing them in the real world, or in simulation. Real-world implementation has had interesting results, such as Adleman's solution of an instance of the directed Hamiltonian path problem using DNA molecules (Adleman, 1994) and Adamatzky's reaction-diffusion logic gates in a chemical medium (Adamatzky and Costello, 2002).

Simulating non-standard computational systems has also been successful; consider Winfree's simulations of computation by interactions between self-assembling tiles (Winfree, 1998) or Adamatzky's simulations of reaction-diffusion systems (Adamatzky, 1997). Winfree simulated sets of DNA molecules with 4 "sticky ends" (ends amenable to binding with other DNA molecules) and showed that they are capable of unsupervised self-assembly, in particular patterns, into nets of DNA. This system was used to simulate a self-assembling Sierpinski triangle. Adamatzky simulated (and also constructed) logic gates whose information carriers were interacting waves of chemical reaction proceeding across a medium.

When simulating chemical computation, we need to set up an environment which approximates real chemistry. As the mechanisms of real chemistry are currently intractable (too much computation is required) and indeed not fully known, we need to set up a simplified simulation-world. It should be qualitatively similar to real chemistry, but vastly simplified to make it computationally tractable.[1] There are several choices we have to make:

[1] Unconventional computation can be a rather recursive field; we are considering the computational power of a simulation whose complexity is limited by the computational power required to simulate it!

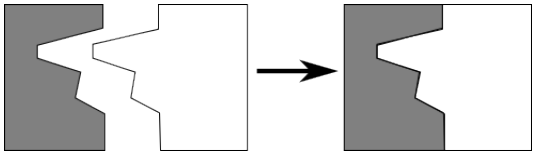

Figure 1: Two "atoms", complementary in shape, bonding together to form a complex.

- *Atoms.* What types of atoms (unsplittable entities) to include in the system.
- *Interactions.* How these atoms interact.
- *Determinism.* How large is the role of chance in the system.
- *Container.* The environment the simulation operates in; its dimensionality and boundary conditions.

Bindworld: a simple simulation and its drawbacks

This section presents Bindworld, a trivial simulated chemical-analogue computational system designed around the concept of binding alone (with no explicit formalisation of reaction). This is a "thought experiment" that shows the necessity of reaction in computational chemistries.

A "program" in Bindworld consists of a population of *atoms*, so called because they are indivisible. Each atom has one or more *bindsites* of type k or k', $k \in \mathbb{N}^*$. For example, a shape could have the three bindsites of types $1, 2', 3$. A bindsite of type k can only bind to one of type k'. For example, a bindsite of type 7 can *only* bind to one of type $7'$. This rule reflects shape complementarity. Only bindsites on different atoms can bond; two complementary bindsites on the same atom cannot bond to each other. If a bond occurs, the two atoms in question are bonded to form a complex.

To implement a program in Bindworld, we must:

1. **INPUT**: Set up a population P_i of atoms, encoded in which are the truth values for our gate inputs. When setting up the population P_i (encoding our question) we have control of the presence or absence of atoms, and of the bindsites they possess. We can choose whether to include a certain atom in the initial population, and which bindsites to equip it with.
2. **COMPUTE**: Let the system evolve by forming all possible bonds between atoms.
3. **OUTPUT**: Interrogate the final population P_f, and infer the state of the system from the pattern of bonds found between atoms.

Bindworld can either be deterministic (if two atoms can bind at a particular moment, they certainly will) or nondeterministic (if two atoms can bind, they may either bind or remain unbound).

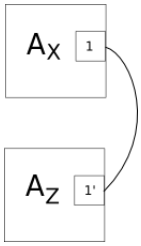

Figure 2: A molecular NOT gate; atom A_X (top) has bound to atom A_Z (bottom) via complementary bindsites of types 1 and 1'. Here the case where X = true is illustrated; the atom A_X is present. The case X = false, this atom would be absent and A_X would be bound to nothing.

One of the implications of having bonding with no reaction is that when reading the final population P_f, it is useless to read either the presence/absence of atoms or which bindsites they possess, as this information will be the same as in the initial population. The only way in which the system can change is by bonding atoms to other atoms and forming complexes. We must therefore extract the output value of our logic gate from information about which atoms are bound to which.

Making logic gates in Bindworld

Suppose we have access to two binary variables X and Z and two atoms A_X and A_Z with complementary bindsites 1 and 1'. These atoms are notated $A_{X,1}$ and $A_{Z,1'}$ and will bond to each other as their bindsites are complementary. We then define the input rule and the output rule formally as:

$$\begin{aligned} \text{Input: } P_i &= \begin{cases} \{A_X, A_Z\} & \text{if } X = T \\ \{A_Z\} & \text{if } X = F \end{cases} \\ \text{Output: } Z &= \begin{cases} F \text{ if} & \text{bound}(A_Z)^{P_f} \\ T \text{ if} & \neg\text{bound}(A_Z)^{P_f} \end{cases} \end{aligned} \tag{1}$$

We set the output variable Z to true if atom A_Z is bound to anything in P_f, shown by the helper function bound$(A_Z)^{P_f}$. This implements the relation $Z = \neg X$ in Bindworld; the chemistry of the system behaves differently depending on how we set up the initial population.

As shown, it is simple to engineer one gate on its own. It is also simple to run a set of gates in parallel by simply using sets of bindsites which do not interact with each other. It is when we come to link gates together by connecting the output of one to the input of the next (as we must to run any meaningful computation) that problems occur.

One way of doing this is with the NOT gate is by simply inserting another atom A_J with a bindsite of type 1. If we do this *after* the initial gate has run, the results will be sensible and $\neg$bound$(A_Z)^{P_f}$ will reflect $\neg\neg X$. However, this means the computer's execution would have to be paused after the operation of every gate, which would make the computer's operation intolerably slow, as such chemical manipulations are very slow. Furthermore, they would need to be controlled either by hand or by an electronic computer; using a traditional computer to facilitate the operation of a novel one (when it could just evaluate the gates electronically) is a waste of resources. We want to be able to program the computer and leave it to run unsupervised.

The process is even more complex when dealing with 2-input gates such as AND or NOR, which raises another disadvantage of Bindworld: the activity of programming it (setting up the atoms and the initial population) is very hard. It is conceptually very nonintuitive and difficult to reason about. This violates another goal in the design of novel computers, that they should be easy to control and program. There may be an arcane, complex way of setting up unsupervised chained gates in Bindworld, involving helper bindsites and ancillary atoms, but it escapes us.

To sum up, Bindworld has several down sides. Firstly, programming it is extremely nonintuitive. Secondly, it seems even simple one-input-gate chaining is very hard unless the population can be adjusted after the evaluation of every gate, a restriction which would cripple the system's power. Thirdly, although we can define the joining of two atoms as a "reaction," Bindworld has no *explicit* concept of reaction, which means it does not integrate very well with our mental schemas of computation and chemistry. The next section describes an artificial chemistry built around this concept.

Computation with the Stringmol artificial chemistry

We give here a brief overview of our molecular system, which is described fully in Hickinbotham et al. (2010a, 2009). A summary of the chemistry is presented below, followed by a description and discussion of molecular structure.

In order to express our observations in a computational system, we identify three major domains. The first defines the underlying *physico-chemical properties* of the atomic components. The second is the coding of the proteins in genes - the *sequence of codes*. The third is the *embodiment* of both the physico-chemical properties and the sequence of codes in the physical world. The physico-chemical properties of the system are immutable, but they specify a space of possible realisations that is immeasurably vast. Genetic systems are specified by the sequence of codes, but importantly, the sequence is *embodied* within the system, thus allowing the enzymes that the sequence encodes to act on the embodiment of the sequence itself and thus modify it. This has places the sequence management apparatus under control of the sequence itself, eventually exploiting the available possibilities that the physico-chemical properties endow upon the embodied system. This phenomenon is the basis of bi-

ological evolutionary systems, where the embodiment of a genetic code in a carrier molecule allows the encoded proteins to "curate" the genome. Initial experiments by Hickinbotham et al. (2010b, 2009) into implementing such a system have resulted in the Stringmol chemistry.

Molecular representation

In Stringmol, the molecular analogues are composed of sequences of token symbols (single letters or symbols such as '`$`' or'`>`') which represent both the structure of the molecule and a series of programmatic instructions. Molecules bind at loci along sequences if there is a match between the sequences at that point. Importantly, the match is inexact, and is modelled as a probability of a bind occurring. The basis of the soft alignment scoring function is based on the scoring method of Smith and Waterman (1981).

Once bound, the two molecules have the potential (by following the programs specified by their strings of instructions) both to create new molecules and to change their composition, thus forming new molecules. This is the reaction component of the system. The sequence is treated like a program, commencing at the beginning of whichever aligned subsequence is furthest from the beginning of its string. There are 7 functional symbols, shown as non-alphabetical characters '`$`', '`>`', '`^`', '`?`', '`=`', '`%`', and '}'. Stringmol uses functional symbols to specify the manipulation of a set of pointers which indicate positions on the molecular strings, and the symbols that the pointers index. During a reaction, alignments are used to specify program flow, commonly acting as place markers and analogues of "goto" statements.

Note that in Stringmol, binding and reaction are completely chronologically and conceptually separate. Once a bind is effected, it remains in place for the duration of the reaction. Another bind cannot interrupt a reaction; a third Stringmol cannot bind to a reaction in progress.

System Architecture

A Stringmol simulation can be considered as a set of reacting molecules whose movements inside a container are governed by a stochastic mixing function. All molecules are subject to *decay* (spontaneous destruction), which places a requirement upon the system to act in order to maintain itself in the face of entropy. Should molecules come sufficiently close to one another, then they can *bind* and subsequent to binding occurring, *react*. The system has a discrete-time clock. At each time step, all the molecules in the system are processed. Actions only occur if energy is available. Energy is consumed via binding and executing each instruction in a reaction; these two events each have an energy cost. The likelihood of binding and the nature of the reaction is encoded in the string of each molecule in the encounter. At one particular time step, we specify that 25 energy units are available. The selection of which events consume the energy is stochastic. The balance between energy availability and the decay rate of the molecule maintains a steady population of molecules. We currently specify that only two molecules can ever participate in a single reaction, and that raw materials for the assembly of new molecules are available in saturation.

Strategy for implementing molecular logic

Our implementation of logic within this artificial chemical system allows us to demonstrate the ability of the system to change state. There are two points arising from this. Firstly, the implementation of the processing is not straightforward, since the reaction-language was not tailored to logic. Secondly, following from the first point, there naturally arises within the system the possibility of evolving the system to deliver fuzzier logical analysis from the initial bootstrap, via a built-in ability of the system to evolve.

We require that the chemical system acts to maintain a population of molecules in an environment where no molecule can persist indefinitely. We thus base our system on a molecular species that we call a *replicase*, R. This molecule R has embodied properties coded into its sequence that allows it to bind to copies of itself and create further copies.

Before an input data molecule D enters the system, the R molecules simply maintain a stable population. The input does not persist in the system, so in order for the system to generate a response that does persist, the input signal must induce two changes in R. In our implementation, R is 'primed' to implement changes to its own sequence when D binds to a specific region, to introduce a signal generating molecule S that not only acts as a replicase, but also generates an output molecule O.

Experiments showed (Hickinbotham et al., 2010b) that changes in the binding site of the replicase allowed new species of replicase to be preferentially copied, and thus drive other replicase molecules to extinction. We exploit this phenomena in the design of our state-change when an input data molecule appears – it changes the sequence of the replicase molecule R it interacts with to always *be copied* rather than *act as the copier*. This means that the original replicase R is swept out of the system, and a new replicase species S takes its place. S not only self-maintains, but also produces *output molecules*.

Designing the molecular species

The sequences of logical Stringmol data input molecules D must perform two tasks. Firstly, they must bind to the replicase molecule, and secondly, they must encode the logical state of the input. For a "true" signal, we specify the sequences `INPUTTTTTT` and `INPUTFFFFF` for false. There are two regions to this molecule. The sequence `INPUT` forms the bind site (shown in yellow in figure 3**II**, where

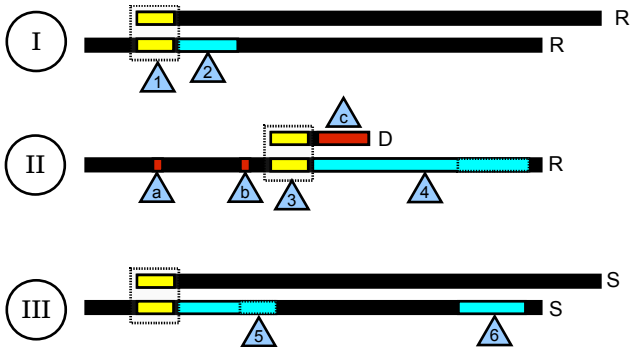

Figure 3: The replicase NOT gate molecules. Yellow regions are bind sites, blue regions are program sequences, red regions are read/write areas external to the program regions. Dotted lines show binds. Black regions are parts of the sequence that are not used in the reaction. Particular regions of molecules are referenced by numbers and letters in triangles. There are three reactions **I**, **II** and **III**: **I:** The replicase R will copy anything that binds at position 1, using the program encoded at region 2. R can bind to other R molecules. **II:** The data molecule D binds to R at region 3. The program at 4 is executed, which changes sites a and b and uses the logic encoding at c to write a specification of the output molecule. This changes the sequence of molecule R to create a new molecule species S. **III:** The signal-producing molecule S. This molecule has the dual functionality of copying the molecule it binds to, and moving program flow from region 5 to region 6, where the output molecule is expressed.

as the sequences `TTTTT` or `FFFFF` encode the logical state (shown in red).

The central challenge in this experiment was designing the replicase molecule R to bind and process the input molecule. The string encoding the functionality for this molecule was 243 symbols long. The reactions that the sequence encodes are shown in figure 3. There are three reactions to consider. Reaction **I** is the replication function, encoded on region 2 the top row of the figure. We refer the reader to Hickinbotham et al. (2009) for a discussion of the replicase functionality encoded in this region. Reaction **II** occurs when D binds to R at region 3. The processing of the input molecule is encoded in region 4, and proceeds as follows:

1. `check-input`: Inexact alignments can form a bind with low but significant probability. It is therefore important to check the validity of molecules which bind at region 3. The sequence `?VACH}` carries out this check, and terminates the reaction if the condition is not met.
2. `Mod-replicase`: This sequence carries out the modification of the replicase bind site at region a, and deletes the terminate symbol } at region b. This change means that region 5 will be executed in reaction **III** to initiate production of the output molecule when S-S binds occur.
3. `Check-boolean`: Switches program flow to create a template for an output molecule that embodies "true" or "false" in the system.
4. `Set-output-false` and `Set-output-true`: These sequences position the read pointer over the sequence encoding "true" or "false" respectively depending on the output of a NOT operation on the logical state of D.
5. `Make-output-message` writes the output of the NOT operation into the template sequence for the output molecule.
6. `Express-output-message` creates a new output molecule. Note this sequence is also shown in reaction **III** as region 6. The program executed by the S jumps to this region from region 5.

The output molecules are `ODTWKZFFFFF` for false and `ODTWKZTTTTT` for true. Note that we had originally coded this molecule using the sequence `OUTPUT`, but the last three letters of this sequence formed a partial match with the bind site for `INPUT`.

It is clear the mechanism for latching the system is more complex than in an electronic logic gate. This is a consequence of the fact that *everything* in the system is subject to decay, so in order to preserve the output, it must be repeatedly created by S. However, this does provide the facility for new configurations to arise by allowing mutation to occur in the system as in (Hickinbotham et al., 2009).

Experimental trial

To demonstrate the effectiveness of the molecular specification, we ran 1,000 trials each of input conditions with true, false and NULL configurations. A previously implemented C++ incarnation of Stringmol was used to run the trials, all of which successfully produced the output signal molecule, maintaining the population indefinitely. Examples of processing a true and false input signal are shown in figure 4 (the null configuration is simply a constant population of the seed replicase). In both of these examples, the molecular population dynamics are similar. The Input signal binds to the Start Replicase, which executes the self-modifying code. This produces the "signal replicase". We see the population of Start Replicase drop off more quickly than the input molecule, since it is subject to modification into the Signal replicase *and* decay, whereas the input molecule is only subject to decay. The bump in the population of the Signal replicase is due to an energy glut, since the input population is too small to consume available energy. Finally, we see the emergence of a steady state population of two molecular species: the Signal replicase, and the output molecule encoded with the appropriate boolean NOT response.

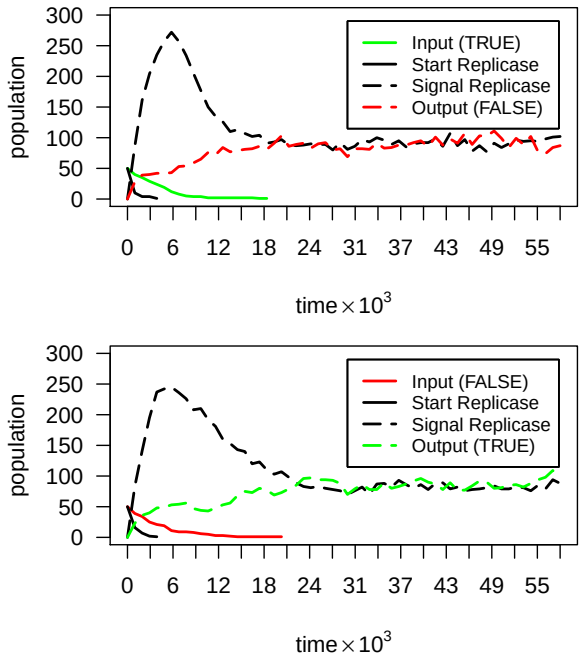

Figure 4: Reaction logic processing of an input molecule through a NOT replicase. **Top**: "true", encoded as `INPUTTTTTT` (green line), when combined with the "start replicase" (black line), creates a population of output "false" molecules encoded as `ODTWKZFFFFF` (red dashed line), along with a Replicase-plus-signal enzyme (black dotted line). **Bottom:** Same as the above, but with "false", encoded as `INPUTFFFFF` (red line), and a population of output "true" molecules encoded as `ODTWKZTTTTT` (green dashed line).

Comparison and conclusion

We have shown theoretically and experimentally how an embodied reaction process enhances the computational power of chemical systems. The main demonstrator for this has been the design of a simple NOT gate. The differences between Bindworld and Stringmol mirror several important considerations in the design and engineering of novel computational systems.

A major difference is that Stringmol is more complex than Bindworld; it has more atoms, more complex combination rules, and a clear, hardwired border between binding and reaction. Informally, Bindworld is a simpler *world* than Stringmol. This means that, to express complex ideas like logic gates in the terms of Bindworld, we have to do more work to reduce them to its simple terms. Stringmol (like a high-level programming language) has more useful abstractions that we can incorporate when designing "code" that runs in Stringmol. Its programmatic instructions encode the reaction potential of each entity. Stringmol is in closer correspondence with our mental schemas of the problems we wish to solve. As ease of programming is a very important computing property, this consideration is important. Stringmol vs. Bindworld is an example of how making a system more complex can make it easier to use.

Stringmol is also nondeterministic; this can be an advantage because it allows the same starting state to cause different behaviours, which may occur at different rates and can aggregate over time into more complex behaviours. Von Neumann computation relies on the permanent assumption of determinism, but this is not necessary (many biological computers, like the brain, do not require it) and means we have to work harder to implement nondeterminism (as in random number generators). It also leads us into a procedural, deterministic, local mental schema of computing, which is not something we want to be crippled by when designing distributed, concurrent or nondeterministic systems.

With artificial chemistries like Stringmol, there naturally arises the possibility of evolving the system to deliver fuzzier logical analysis from the initial bootstrap, via a built-in ability of the system to evolve. We plan to explore this avenue in future work, with a view to delivering a system capable of evolving solutions within the embodied chemistry.

In our experimental work, we have taken pains to develop a solution that required no changes to the artificial chemistry that was used in (Hickinbotham et al., 2010b) for applications in evolution. This is important, since biological evolution exploits the embodiment of the genome in much the same way as the embodied reactions we explore here. We had to overcome some difficulties with the functional codes in Stringmol, and also some difficulties with similarities in binding sequences that led to mis-alignments. These indicate that the Stringmol system would have more expressive power in both evolutionary and computational experiments were these issues addressed.

Rather like Newspeak in Orwell's 1984, the expression of certain things in Stringmol is difficult if not well-nigh impossible. Intuitively, the process of simulating the physical copying of a sequence of letters seems to require more information processing than a single logic gate. However the process of copying information is not the same as actually processing the information itself. Thus in Stringmol it is easy to copy strings, but it is very difficult to express a straightforward logic gate because there are not the straightforward expressions available to do so.

We note that simple changes to the Stringmol language that would emphasise the concept of molecular embodiment of information would make the logic gate easier to implement. These are: • **Cut and paste** of strings, rather than copy and paste. Currently, we have to laboriously copy each symbol in a string to do some information processing. But some of these operations do not really require copies to be made. We could just as easily use what is being copied.

Thought of as an embodied system, the advantages of this are clear. Cut and paste in Stringmol would mean that subsequences could be excised and spliced into other sequences by manipulation of pointers. • **Regulation** of some behaviour could occur if repressors could be made possible. In our NOT gate, the input signal re-programmed the seed replicase so that it made an output signal. This programming would not have been necessary in a system where regulation of expression was feasible. • **Energy-dependent behaviour:** Our processing system is subject to energy constraints. If we could switch when energy was low, we could change behaviour to correct it. This would allow regulation of energy to occur and give rise to selection at the molecular level that is not currently possible • **More steps to copy:** If we could dismantle the '=' operator, we'd be able to do more sophisticated construction of signals. As it is, we have to string ====== together to copy short sequences, that is not obviously evolvable without six corresponding mutations. If we could use the Nellis-Stepney system, we could increment the write pointer without incrementing the read pointer, and thus have a copy of a symbol many times. • **Increment direction:** If we could switch this, it would be possible to write/evolve more compact programs. • **Pointer referencing:** If we could move *any* pointer to any other pointer, rather than the limited set currently implemented, we could more easily implement certain information processing behaviours. probable alignments.

It is interesting to note that many of these extensions could be applied to other string-based ALife systems, such as Tierra (Ray, 1991), Avida (Johnson and Wilke, 2004) and Typogenetics (Gwak and Wee, 2007). These systems were also designed to carry out the task of replication, and they are known to have limitations in terms of evolutionary open-endedness. Our studies here indicate that there is potential to extend the instruction sets in these models to allow richer information processing, which may lead to richer behaviours.

Stringmol and Bindworld are doubtless far from any useful chemical computer, being after all only simulations. However, they allow us to explore the properties of chemical and bio-inspired long molecule computing, a strategy which we hope will eventually allow us to design a useful biological computer.

Acknowledgements

The authors thank Susan Stepney, Peter Young, Tim Clarke, Edward Clark and Adam Nellis for comments and suggestions during the preparation of this manuscript. Simon Hickinbotham is funded by the Plazzmid project, EPSRC grant EP/F031033/1.

References

A. Adamatzky and B. D. L. Costello. Experimental logical gates in a reaction-diffusion medium: The XOR gate and beyond. *Physical Review E*, 66(4):046112.1–046112.6, 2002.

A. I. Adamatzky. Universal computation in excitable media: the 2+ medium. *Advanced materials for optics and electronics*, 7 (5):263–272, 1997. ISSN 1099-0712.

L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

W. Aspray. John von neumann and the origins of modern computing. *Technology and Culture*, 33(1), 1990.

R. A. Brooks. Steps towards living machines. In *ER 2001*, pages 72–93. Springer-Verlag, 2001.

E. Clark, S. J. Hickinbotham, S. Stepney, T. Clarke, A. Nellis, M. Pay, and J. P. Young. Degeneracy enriches artificial chemistry binding systems. In *ECAL 2011*. MIT Press, 2011.

B. J. Copeland. The Church-Turing thesis. *Stanford encyclopedia of philosophy*, pages 1095–5054, 1996.

E. A. Di Paolo, J. Noble, and S. Bullock. Simulation models as opaque thought experiments. In *Alife VII*, pages 497–506. MIT Press, 2000.

W. Gilbert. Origin of life: The RNA world. *Nature*, 319(6055): 618, February 1986. doi: 10.1038/319618a0.

C. Gwak and K. Wee. Construction of hypercycles in typogenetics with evolutionary algorithms. In *ECAL 2007*, pages 1060–1068. Springer-Verlag, 2007.

S. J. Hickinbotham, E. Clark, S. Stepney, T. Clarke, A. Nellis, M. Pay, and J. P. Young. Molecular microprograms. In *ECAL 2009*, LNCS, pages 291–298. Springer, 2009.

S. J. Hickinbotham, E. Clark, S. Stepney, T. Clarke, A. Nellis, M. Pay, and J. P. Young. Specification of the stringmol chemical programming language version 0.1. Technical Report YCS-2010-458, Univ. of York, 2010a.

S. J. Hickinbotham, A. Faulconbridge, and A. Nellis. The blind watchmaker's workshop: Three artificial chemistries in the context of eigen's paradox. In *ALife XII*. MIT Press, 2010b.

T. J. Johnson and C. O. Wilke. Evolution of resource competition between mutually dependent digital organisms. *Artif. Life*, 10:145–156, April 2004.

J. Y. Lee, S. Y. Shin, T. H. Park, and B. T. Zhang. Solving travelling salesman problems with DNA molecules encoding numerical values. *BioSystems*, 78(1-3):39–47, 2004.

G. Păun, G. Rozenberg, and A. Salomaa. *DNA computing: new computing paradigms*. Springer Verlag, 1998.

T. S. Ray. An approach to the synthesis of life. In *ALife II*, pages 371–408. Addison-Wesley, 1991.

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–197, 1981.

M. N. Stojanovic and D. Stefanovic. A deoxyribozyme-based molecular automaton. *Nature Biotechnology*, 21(9):1069–1074, 2003.

C. Teuscher. Exploring logic artificial chemistries: An illogical attempt? *CoRR*, abs/cs/0703149, 2007.

E. Winfree. Simulations of computing by self-assembly. *DIMACS: DNA-Based Computers*, pages 213–242, 1998.