Logically and Physically Reversible Natural Computing: A Tutorial

Chris Thachuk[*]

Department of Computer Science, University of Oxford, Oxford, UK
chris.thachuk@cs.ox.ac.uk

Abstract. This year marks the 40th anniversary of Charles Bennett's seminal paper on reversible computing. Bennett's contribution is remembered as one of the first to demonstrate how any deterministic computation can be simulated by a logically reversible Turing machine. Perhaps less remembered is that the same paper suggests the use of nucleic acids to realise physical reversibility. In context, Bennett's foresight predates Leonard Adleman's famous experiments to solve instances of the Hamiltonian path problem using strands of DNA — a landmark date for the field of natural computing — by more than twenty years. The ensuing time has seen active research in both reversible computing and natural computing that has been, for the most part, unrelated. Encouraged by new, experimentally viable DNA computing models, there is a resurgent interest in logically reversible computing by the natural computing community. We survey these recent results, and their underlying ideas, which demonstrate the potential for logically and physically reversible computation using nucleic acids.

Keywords: reversible computing, natural computing.

1 Introduction

By the 1960's, scientists and mathematicians concerned with the study of computing had already begun to ask and answer the question: what can be computed *efficiently*? Many results emerged showing that seemingly difficult problems could be solved by algorithms that had time complexity bounded by a polynomial — a criterion Edmonds advocated as a measure of a (time) efficient algorithm [7]. We can extend this question to ask: what constitutes an *energy* efficient algorithm? More generally, can any computation be performed in an energy efficient manner? By 1961, this question was partially answered when Landauer proved that it was only logically irreversible operations — those which cause information loss — that must expend energy [12]. Unfortunately, deterministic computation is not necessarily logically reversible and typical programming is unlikely to be so. Fortunately, it was later shown that any deterministic computation could be simulated by a logically reversible Turing machine, thus showing that computation does not, in principle, have a fundamental limit with respect to energy expenditure. The result emerged independently by Lecerf [14] in 1963 and later by Bennett [2]

[*] This work is supported by the ERC Advanced Grant VERIWARE and by the Oxford Martin School.

in 1973; although Lecerf's result was little known and Bennett's is often cited as the seminal paper for logically reversible Turing machines.

While Turing machine models can be used to reason about theoretical improvements, any reaped benefit of energy efficient computation must occur in the physical world. Thus, one must also consider computing with physically reversible systems, whether they be electronic or quantum circuits, billiard-ball computers [8], or an altogether different physical system. In his seminal paper, Bennett suggests using the standard machinery of a cell for the bio-synthesis and bio-degradation of messenger RNA — a nucleic acid similar to DNA — as a means for physically reversible computation, where the synthesis and degradation actions are analogous to reading and writing, respectively, to a Turing machine tape. Interestingly, this idea predates the demonstrated use of nucleic acids for computation by more than twenty years [1], and the field of DNA nanotechnology in general, by nearly a decade [24].

Using just four bases (A,C,G, and T), DNA acts as a storage device by encoding genes and other blueprint sequences that can be inherited by future generations. This purpose of DNA is very much in line with Bennett's original insight. However, DNA is not limited to the role of information carrier. Consider the most common shape associated with DNA — the famous double helix. This structure is formed by two sequences, in opposite orientation,[1] that *hybridize* together by forming bonds between complementary bases (see Fig. 1 (top left)). The A base will bond with a T base and, similarly, C will bond with G. The beginning of DNA nanotechnology is largely attributed to a paper by Seeman [24] in 1982 where he demonstrated the potential for DNA to assume shapes other than the double helical structure. This is accomplished by a careful design of strands and, in particular, by a careful design of *domains*, or subsequences, of those strands so that when they are added into the same solution, they *self-assemble*, via hybridization, into the intended shape (see Fig. 1 (top right)). DNA has since proven itself to be an effective and programmable construction material for engineering arbitrary shapes at the nanoscale [19].

In addition to self-assembly into static structures, DNA hybridization can be leveraged for creating dynamic systems that change over time. This has led to the exploration of using DNA to perform computation. The advantage? A natural interface with biological systems that can be implemented *in vitro* and, potentially, *in vivo*. Many models of computing with DNA have arisen over the years, including the *Adleman-Lipton* model — based on the ideas underlying Adleman's famous experiments to solve instances of the Hamiltonian path problem [1], the *Sticker* model [22] — where short DNA molecules 'stick' and 'unstick' to a long template strand, analogous to a Turing machine tape, and the *Tile self-assembly* model [27] where 'tiles' of DNA containing different types of 'glue' on each side can hybridize together and give rise to periodic shapes such as the Sierpinski triangle [20].

In the remainder of this tutorial, we limit our focus to one natural computing model that overlaps with the goals of reversible computing. In particular, we concentrate on a relatively new DNA computing model using so-called *DNA strand displacement* systems (DSDs) that provide a natural mechanism to perform physically reversible

[1] A *strand* of DNA is oriented and has a 5' end and a 3' end. Hybridization can only occur between two complementary sequences of DNA in opposite orientation.

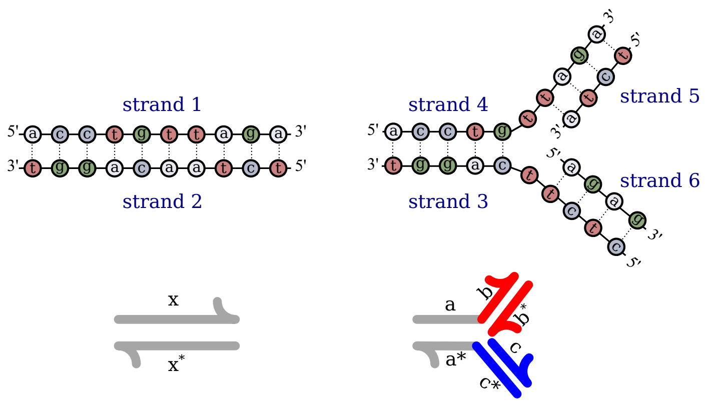


Fig. 1. Single stranded DNA molecules, or *strands*, are polymers over four monomer units called bases. The strands are oriented and have a 5' end and a 3' end. A sequence of DNA, or substrand, is complementary to another if their bases are complementary and in opposite orientation. An A base is complementary to the T base, as the C base is to the G base. When single stranded DNA molecules are added in solution, complementary sequences will hybridize together to form stable double stranded structures. For example, when strand 1 and strand 2 are added in the same solution, a DNA duplex forms (top left). Similarly, when strands 3,4,5 and 6 are added in the same solution, they form a branched structure (top right). When designing strands to self-assemble into different shapes, it is common to abstract their sequences into labeled *domains* which are used to indicate complementary sequences (bottom).

computation steps. We focus on DSD systems as they are simple, widely studied and experimentally practical. These systems leverage the fact that an unbound strand A can still hybridize with a complementary domain on some strand B, even if it is already hybridized to some other strand C. If A does hybridize to B, strand C is said to be *displaced* and can next be used to displace some other strand [28]. DNA strand displacement mechanisms have been experimentally implemented and verified to simulate neural networks [18], Boolean logic circuits [23,5], and even reversible Boolean logic circuits [9], among numerous other applications. As we will see, they are also capable, in principle, of physically and logically reversible Turing-universal computation [16,11].

2 Background

In the natural computing results we study in this tutorial, a distinct notion of logically reversible computation is used that differs from the standard definition. We begin with a discussion of this distinction in terms of configuration graphs, as opposed to restrictions on Turing machine transitions [2], which will be used to simplify our presentation of natural computing examples. This is followed by an overview of the DNA strand displacement model, and stochastic chemical reaction networks.

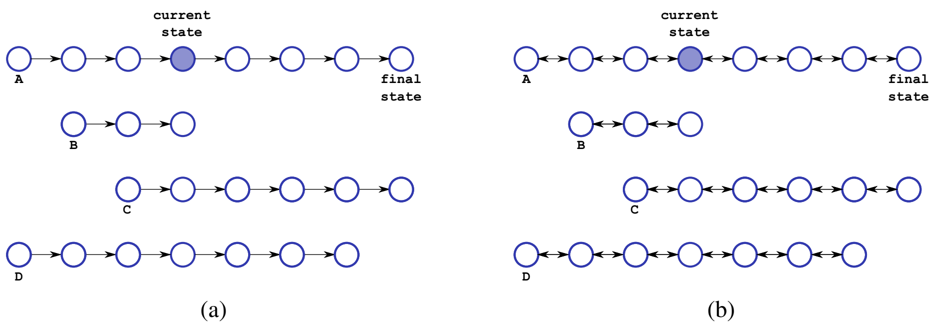


Fig. 2. Example configuration graphs, induced on four different inputs, for (a) logically reversible computation, and (b) logically reversible computation (with symmetric transitions) arising in some chemical reaction networks. Nodes represent possible states in a computation and directed edges denote valid state transitions.

2.1 Logical Reversibility

The reversible computing results we will discuss in this tutorial use a slightly different notion of logical reversibility. As we will see, this distinction is important as the physical reversibility of these systems is actively exploited to achieve space efficient computation. For our purposes here, it suffices to understand the important difference distinguishing this notion from the standard notion of logically reversible computation. The intuition of the difference is captured well by considering the *configuration graph* of a computation which has a node for every possible state on every possible input for the underlying Turing machine being modeled. There is a directed edge from node i to node j if and only if state j is reachable from state i in a single state-transition of the Turing machine.

Shown in Fig. 2a is a configuration graph of a typical logically reversible computation, shown for four different inputs (source nodes A-D). Importantly, a logically reversible computation for a particular input forms a directed path, from its input to its final state, which is unconnected to any state for any other possible input. This means any state along the chain can be deterministically reached from the final state, if the entire chain is reversed. Thus, information is *not* lost. Contrast this with the configuration graph of a logically reversible computation (with symmetric transitions) for the same four inputs shown in Fig. 2b. The only difference is that each non-terminal node along the chain has two possible choices of where to next proceed: its successor state, or its predecessor state. Therefore, each transition, and the overall computation, is *symmetric* [15]. As we will see, the state transitions in the systems we consider model reversible chemical reactions. That is, after each reaction it is possible the reaction is immediately reversed. The computation is still logically reversible in the following sense. One choice is always the previous state of the computation. (Retreating to the previous state is equivalent to the transition never having occurred.) The important point is that at any given node, the computation cannot proceed to more than one other node that is not the previous state.

2.2 DNA Strand Displacement Systems (DSD)

A DNA strand displacement system (DSD) consists of *unbound* strands and double stranded complexes consisting of one or more bound strands to a long *template* strand. For example, in Fig. 3a there is one unbound strand labeled I and one double stranded complex consisting of the strand labeled E bound to the template strand labeled S. Strands in the system are composed of two types of strand domains: short *toehold* domains, and *long* domains. Distinct domains are assumed to have a distinct sequence design. The short length of toehold domains is chosen to ensure strands bound together only by a toehold can spontaneously unbind from one another at the experimental temperature. In contrast, two strands bound by a complementary long domain is considered a stable binding such that that they cannot spontaneously unbind at the experimental temperature.

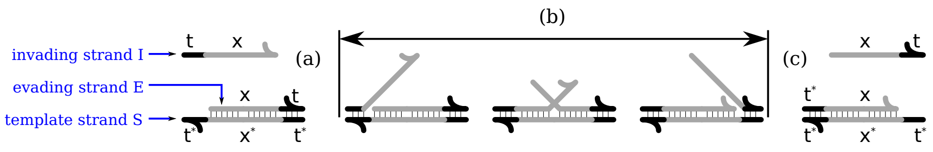


Fig. 3. A successful toehold mediated strand displacement. (a) Toehold t (black subsequence) of the invading strand I binds with its unpaired complement, t^*, on the template strand S. (b) The invading and evading strands share the common long domain x (gray subsequence). They compete, via a random walk process, to bind with the complementary long domain x^* of the template strand until all bases of the invading strand are paired. (c) Toehold of the evading strand E detaches from the template strand S, at which point it has been displaced. Since there remains a free toehold on S, the process is reversible as strand E could displace strand I.

The fundamental operation in a DSD is strand displacement, whereby a toehold domain of an unbound strand, called the invading-strand, binds to an unbound complementary toehold domain of a template strand and, if the adjacent long domain is complementary, it can displace a currently bound signal strand, called the evading strand, of the same length. We illustrate a simple, reversible version of strand displacement in Fig. 3. First, the toehold t of the invading strand, I, binds (forms base-pairs) to the complementary toehold t^* of the template strand, S. Note that any strand with toehold t could initially bind here. However, the process only continues if the adjacent long domain of invading strand I is identical to the long domain of evading strand E. If I and E do share an identical long domain, then in random walk process (often referred to as three-way branch migration), the bases of the long domain of I compete with those belonging to the identical long domain of E to form base pairs with the complementary long domain of the template strand S. Once the long domain of I has bound to its complement domain on the template strand, strand E remains bound by just its short toehold domain. Due to their short length, the toehold bonds can break, thereby releasing signal E. (Of course I may detach from the template before E is released, in which case the displacement does not happen.) The displacement is physically reversible because signal E can next bind to the template strand S to displace strand I

via the same principles. However, the template complex does have an orientation. For example, after I has displaced E in the original displacement, another strand identical to I cannot be used to displace another strand identical to E. This will only be possible, using this same template complex, after some strand E displaces the bound strand I. Alternatively, an I strand can displace and E strand if a another copy of the template complex is present in its original orientation.

2.3 Chemical Reaction Networks (CRN)

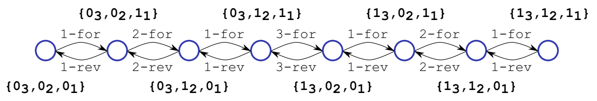
Just as a DSD abstracts sequence level details of interacting DNA strands using the concept of domains, stochastic chemical reaction networks (CRN) abstract details about displacements. CRNs provide a concise language for writing molecular programs and affords us the opportunity to express complex ideas more succinctly. A chemical reaction equation details a process whereby certain molecule types can be consumed — the reactants — and others produced — the products — within some reaction volume. A reaction may also require the presence of catalyst molecules of certain types. A catalyst molecule is neither consumed nor produced by a reaction, but rather it facilitates a reaction which could not otherwise occur without its presence. We refer to all three categories of molecules, generically, as signal molecules. For example, the reaction $A + B \overset{C}{\to} D$ consumes a signal of type A and a signal of type B and produces a signal of type D in the presence of the catalyst[2] signal C. This is an example of an irreversible reaction; however, $A + B \overset{C}{\rightleftharpoons} D$ is an example of a reversible reaction meaning that both a signal of type A and of type B can also be produced by consuming a signal of type D in the presence of the catalyst signal C. A CRN is a set of chemical reactions, in addition to a multiset of signals present within the reaction volume, prior to any reaction occurring, called the initial signal multiset. The current signal multiset is the current composition of signals of a given CRN within a reaction volume — in terms of computation, this specifies the state of the system. From a given state, any reaction can be applied if both the required reactants and catalysts are in the current signal multiset. Thus, arbitrary CRNs are not necessarily deterministic. Importantly, it has been shown that any chemical reaction equation can be realized by a physically reversible DNA strand displacement cascade [16,3].

Let us consider a concrete example of a 3-bit standard binary counter that should begin at count 000, advance to 001, and so on, until reaching the count 111. In our molecular program, we let signal 0_i and signal 1_i denote that bit i has value 0 and 1, respectively, for $1 \leq i \leq 3$. Thus, our 3-bit counter will have the following initial signal multiset: $\{0_3, 0_2, 0_1\}$. Fig. 4a gives three chemical reaction equations for exchanging signals and thus changing the state, or current signal multiset, of the counter. Initially, only reaction 1 can be applied in the forward direction. This is because all other reactions require at least one of the bits to be set to 1. When reaction 1 occurs, signal 0_1 is consumed and signal 1_1 is produced, putting the counter in state $\{0_3, 0_2, 1_1\}$.

[2] Some reactions require the presence of one or more signals, called *catalysts*, which they do not consume. Note how we represent catalysts in our reaction equations. These are not to be confused with rate constants which do not factor into our current discussion. Catalysts do play a significant role and this representation was chosen for its succinctness.

$$\begin{array}{lll} (1) & 0_1 & \rightleftharpoons 1_1 \\ (2) & 0_2 + 1_1 & \rightleftharpoons 1_2 + 0_1 \\ (3) & 0_3 + 1_2 + 1_1 & \rightleftharpoons 1_3 + 0_2 + 0_1 \end{array}$$

(a)



(b)

Fig. 4. (a) Chemical reaction equations for a 3-bit standard binary counter. (b) The configuration graph of the computation performed by the 3-bit standard binary counter forms a chain and is logically reversible (with symmetric transitions). The nodes represent the state of the computation and the edges are directed between states reachable by a single reaction.

From this state, two reactions are next possible. Either the previous reaction is reversed (i.e., reaction 1 next occurs in the reverse direction), or reaction 2 can occur in the forward direction, resulting in state $\{0_3, 1_2, 0_1\}$. Similarly, either reaction 2 is reversed, or reaction 1 next occurs once again, bringing the counter to state $\{0_3, 1_2, 1_1\}$. This continues until the counter reaches the final state $\{1_3, 1_2, 1_1\}$. Fig. 4b represents all reachable states of the counter as nodes and has edges between states that are reachable within one reaction step. Notice that this CRN specifies a logically reversible computation (with symmetric transitions). This 3-bit counter can be extended to a 4-bit counter by adding signal 0_4 to the initial signal multiset, and by adding the reversible reaction $0_4 + 1_3 + 1_2 + 1_1 \rightleftharpoons 1_4 + 0_3 + 0_2 + 0_1$. In a similar manner, the CRN can be extended to simulate any n-digit counter.

3 Reversible and Turing-Complete Natural Computing

Are DNA strand displacement systems capable of logically and physically reversible Turing-universal computation? This question was answered in the affirmative by Qian *et al.* [16] whose work stands as one of the most important theoretical contributions to the area. In a first major contribution of that work, the authors offer a design for a reversible DSD reaction cascade that can realise any chemical reaction. An example for the reversible reaction $A + B \rightleftharpoons C + D$ is given in Fig. 5. The signal molecules are represented by the strands in the shaded boxes, consisting of three domains. The other strands and complexes, which facilitate the reaction, are collectively called the reaction *transformer*. Note that for a reversible reaction cascade, the transformer has two orientations — one for each direction of the reaction. In Fig. 5, the forward orientation is shown top-to-bottom, where the DNA strands for the signals A and B are consumed, and those for C and D are produced. The reverse is shown from bottom-to-top. Consider the forward reaction (top-to-bottom) in Fig. 5. Initially, the only available toehold complement on the template strand is adjacent to a domain complementary to a long domain of strand A. Once strand A displaces a bound strand, a new toehold is available for strand B to bind. Next, auxiliary strands are used to displace C and D — the intended products of this reaction. Once the transformer is in this orientation, it cannot be used to consume A and B and produce C and D. It can however perform the reverse reaction next.

In their second major contribution, the authors enriched the DSD computing model by showing how displacement reactions could be used to add and remove strands to

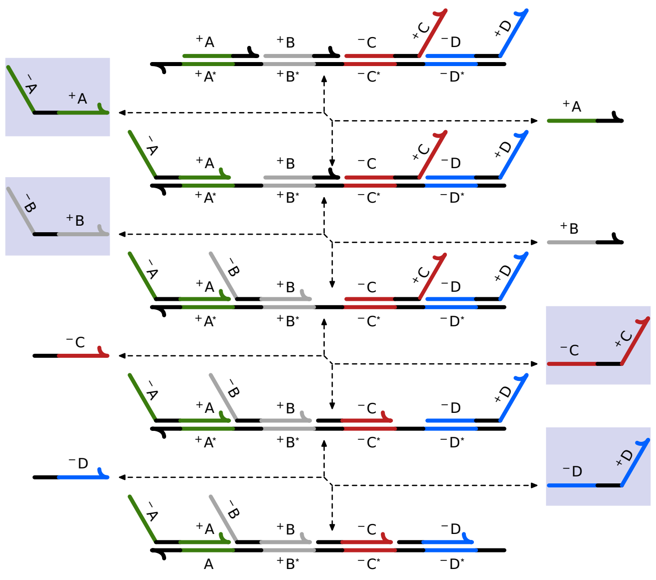


Fig. 5. A strand displacement implementation of the bi-molecular chemical reaction equation $A + B \rightleftharpoons C + D$ using the construction proposed by Qian *et al.* [16]

a growing polymer. Consider the example in Fig. 6, clockwise from state 1 to state 4, showing how a signal denoting the value X can be appended to a growing polymer. Initially, in state 1, the polymer is considered to be an empty list and consists of a single strand, with one toehold and one long domain, that denotes the head of the list. In state 2, the domains denoting the head can interact with a transformer, containing the value X, to form a new extended complex. However, X is only tentatively appended. The process can continue into state 3 if a signal strand, denoting that value X is present, interacts with the new extended complex. Finally, the process in finalized in state 4 when the new extended complex interacts with another transformer resulting in the original exposed domains denoting the head of the list. At this point a new value could be appended to the list, or the previous value X could be removed by performing the reaction sequence in reverse.

Coupled with a clever design of reactions to control state, the authors were able to simulate a (multi) stack machine using only strands of DNA. As the stack machine could be used to simulate Bennett's original reversible Turing machine, and since the reaction cascades of the stack machine simulation are reversible, the authors demonstrated that Turing-universal computation could be realized by a logically and physically reversible DSD sytem enriched with polymers.

One important issue must not be overlooked. As Bennett points out [2], a physical system performing a computation of length t, with no positive drift in the forward direction, will reach the end state in t^2 expected steps. However this should not be considered a computation, as the process can immediately reverse once reaching the output state — the probability of observing the output in this manner is only $\frac{1}{t}$. To overcome

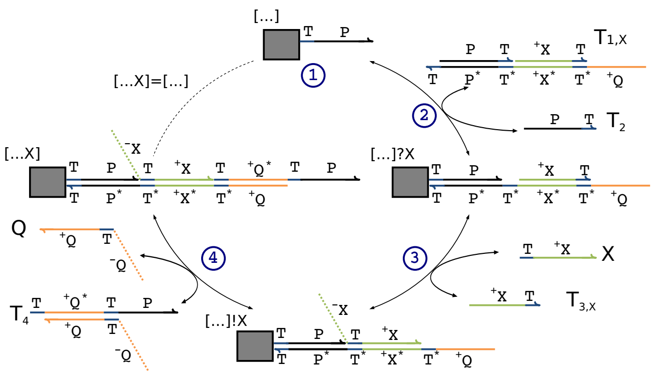


Fig. 6. A reproduction of a figure from Qian *et al.* [16] illustrating (clockwise from the top) how a new signal X can be *pushed* on to a stack by a reaction cascade for polymer extension

this in the stack machine construction, the authors introduce a small bias in favour of forward reactions, thus introducing a positive drift in the computation chain towards the output state. The positive drift is accomplished as follows. DSD reactions are always bi-molecular reactions between two distinct species types (one invading strand and one transformer complex). The propensity of a reaction $A + B \to \dots$ within a well-mixed solution of size v is $\frac{|A||B|}{v}$, where $|X|$ is the count of molecules of type X. Consider the reaction $X + T_f \rightleftharpoons Y + T_r$ and suppose T_f is the transformer for the forward reaction, and T_r is the transformer for the reverse. The propensity of the forward reaction can be made to be twice as large as the propensity of the reverse by ensuring $|T_F| = 2|T_R|$. Thus, in the stack machine construction, the authors ensure more copies of each reaction transformer is in the forward orientation, rather than the reverse, for the duration of the computation. To next reverse the computation towards the beginning state, additional transformers in the reverse orientation could be added to the system to bias the computation to next reverse.

4 Reversible and Space-Efficient Natural Computing

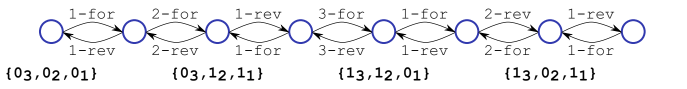
As with Bennett's original reversible Turing machine, the stack machine construction uses space proportional to the length of the computation. This is because a new copy

(1) $0_1 \rightleftharpoons 1_1$

(2) $0_2 \overset{1_1}{\rightleftharpoons} 1_2$

(3) $0_3 \overset{1_2+0_1}{\rightleftharpoons} 1_3$

(a)


{0₃,0₂,1₁}
{0₃,1₂,0₁}
{1₃,1₂,1₁}
{1₃,0₂,0₁}
1-for 2-for 1-rev 3-for 1-for 2-rev 1-rev
1-rev 2-rev 1-for 3-rev 1-rev 2-for 1-for
{0₃,0₂,0₁}
{0₃,1₂,1₁}
{1₃,1₂,0₁}
{1₃,0₂,1₁}

(b)

Fig. 7. (a) Chemical reaction equations for a 3-bit binary reflecting Gray code counter. (b) The configuration graph of the computation performed by the 3-bit binary reflecting Gray code counter forms a chain and is logically reversible (with symmetric transitions).

of a transformer must be used for every computation step as all reactions of the stack machine simulation, from the beginning state to the ending state, are in the forward direction. Condon *et al.* [6] asked if space-efficient reversible computation is possible in CRNs and DSDs. They showed that, in principle, it was by demonstrating how transformers can be *actively* recycled during a computation. Specifically, the authors proposed an n-bit binary counter that can perform a logically reversible computation through 2^n states, using only $poly(n)$ strands in a DSD system. Their counter was based on the binary reflecting Gray code (BRGC) sequence. Due to the symmetric nature of that particular sequence, only one copy of each reaction transformer is necessary to complete the computation, as each particular reaction is performed alternately in the forward and reverse direction. An example of a 3-bit BRGC counter is given in Fig. 7. As with the standard binary counter, only reactions advancing the counter to a successor state, or to the predecessor state is ever possible.

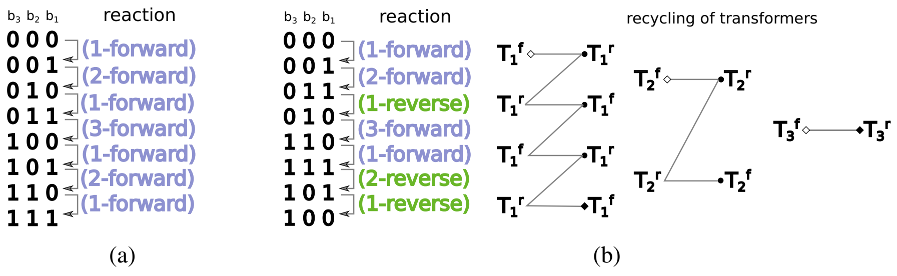


Fig. 8. Comparing two different binary counters over 3 bits. (a) To reach the end state, the standard binary counter must perform a sequence of reactions that always occur in the forward direction, thus requiring a new transformer for every reaction as they are not recycled. (b) The binary reflecting Gray code counter only requires one transformer per reaction equation as each reaction occurs alternately in the forward and the reverse direction.

Contrast the sequence of reactions required for the standard counter in Fig. 8a with that of the Gray counter in Fig. 8b. A standard n-bit binary counter, much like the stack machine, performs only forward reactions when progressing in a computation and would therefore require $2^n - 1$ transformers to reach the end state. In the case of the 3-bit counter, seven transformers are required in total for the standard counter to reach the end state. However, the Gray counter has a regular symmetry which can be exploited to use only one transformer per bit of the counter. In the case of the 3-bit counter, only three transformers are required to reach the end state. In general, the Gray counter is exponentially more space-efficient than the standard counter.

4.1 Technique: Active Computation Reversal

Interestingly, the recursive nature of the binary reflecting Gray code sequence leads to a powerful technique for logically reversible computing. To extend the 3-bit BRGC counter to a 4-bit BRGC counter, all that is required is to add the signal 0_4 to the initial

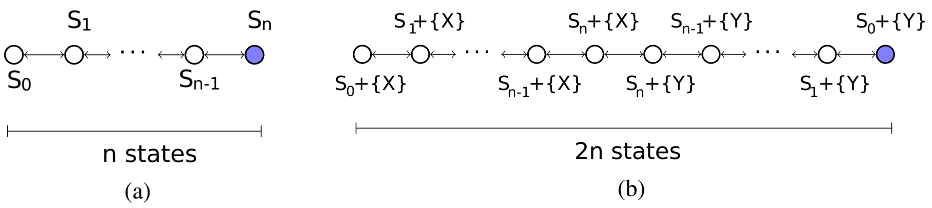


Fig. 9. (a) The computation chain of a logically reversible CRN (with symmetric transitions). (b) The computation chain from (a) doubled by adding one new initial signal, X, and one new reaction, $X \overset{S_n}{\rightleftharpoons} Y$, that requires the signals of the previous final state as catalysts.

signal multiset and to add the new reaction $0_4 \overset{1_3+0_2+0_1}{\rightleftharpoons} 1_4$. The new signal molecule does not affect the reactions for the first 3 bits; furthermore, the new reaction requires as catalysts the signals of the final state of the 3-bit counter. This means the original reaction sequence of the 3-bit counter will proceed prior to the new reaction. Once the 3-bit sequence is complete, the new reaction can occur for the first time to produce 1_4. Other than next reversing this new reaction — stepping back in the computation chain — the only other possibility is to perform the entire reaction sequence of the 3-bit counter in reverse. Note that since the 1_4 signal was not present before, this means the computation is actually stepping forward towards a new end state. By introducing one new signal, and one new reaction, we can effectively double the computation chain length, and ensure active recycling of transformers. The general technique is illustrated in Fig. 9.

When using active recycling of transformers the computation chain must be unbiased as reactions are repeated in alternating orientations — thus we cannot force a positive drift as in the stack machine construction. To ensure the output of the computation can be witnessed with high probability, this same technique can be used to repeatedly extend the overall computation chain. For instance, consider a computation for a decision problem where a special signal is produced to indicate if the input is accepted and another signal is produced to indicate if it is not. This extension technique can be used to double the overall computation chain length, ensuring the output signal can be observed in strictly more than half of the states. As the computation performs an unbiased random walk along the logically reversible computation state space, the steady state probability of observing the output signal is $p > 0.5$. In this manner, for every new reaction added to the CRN to double the chain length, the probability of not observing an output signal is cut in half. Formally, the probability of observing an answer becomes $p'' > 1 - 2^{-(1+c)}$ when $c \geq 0$ number of new reactions are added to extend the computation chain. Thus, we can make the steady state probability of observing a solution signal arbitrarily high.

This same chain extension technique was used by Thachuk & Condon who gave a space-efficient and logically reversible CRN for performing an in-order traversal of a complete binary tree. Coupled with the ideas for verifying a 3-SAT formula (which we explore next), and the new tree traversal procedure, the authors demonstrated how any *quantified* 3-SAT problem instance could be solved, by giving a logically reversible

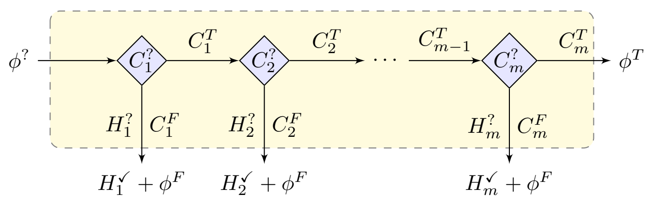
QSAT solver that could be realized as a physically reversible DSD [26]. Thus, any problem in PSPACE can be solved by a space-efficient and reversible DSD. This result was later generalized to solve any problem in SPACE — the class of all space bounded computation [25].

4.2 Designing a 3-SAT Solver

In this section, we will explore, at a high level, how the logically reversible and space-efficient BRGC counter can be used in conjunction with a 3-SAT verification procedure to solve 3-SAT instances. We will use a very simple strategy. For a formula with n variables, an n-bit counter can enumerate every possible variable assignment, and a verification procedure can be run every time the counter changes. The entire computation can be made to halt either when a satisfying solution is found, or when all states of the counter have been exhausted. Furthermore, the entire computation chain will be logically reversible. We do not discuss the specifics of how the counter and the verification procedure can be coupled, but a set of auxiliary reactions are sufficient to achieve the desired result [25]. We note that one detail which cannot be overlooked in such a coupling is that the entire verification procedure must be reversed in between invocations for different variable assignments. Next, we discuss how a 3-SAT formula can be verified. The procedure illustrates the use of history signals to ensure the overall procedure is logically reversible.

$$
\begin{array}{ccc}
C_i^? & \overset{x_j^F + x_k^F + x_l^F}{\rightleftharpoons} & C_i^T \\
C_i^? & \overset{x_j^F + x_k^F + x_l^T}{\rightleftharpoons} & C_i^T \\
C_i^? & \overset{x_j^F + x_k^T + x_l^F}{\rightleftharpoons} & C_i^F \\
 & \vdots &
\end{array}
$$

(a)



(b)

Fig. 10. (a) A set of reactions acting as a *truth table* for clause i. (b) Flow control when verifying a formula ϕ having m clauses.

Verifying an Arbitrary Clause. Consider an arbitrary 3-SAT clause over three literals, each for a distinct variable. There are exactly eight possible truth assignments to the three distinct variables. Thus, for each such clause, we create a set of eight reversible reactions to determine if the clause is satisfied. The reactions for verifying the i^{th} clause, containing literals for variables x_j, x_k and x_l are given in Fig. 10a. When the clause signal molecule $C_i^?$ is present, exactly one of the eight reactions can be applied, specified by the current variable assignment. The variable signals act as catalysts and the $C_i^?$ signal is consumed producing either a C_i^T signal if the clause is satisfied, or C_i^F otherwise. In this example, it is supposed that the variable assignment $x_j = F, x_k = T, x_l = F$ does not satisfy the clause, while the first two variable assignments do. Note that for a particular variable assignment, only one reaction will apply in both the forward and reverse direction, ensuring the process is logically reversible.

Verifying the Overall Formula. For the formula to be true, all clauses must be satisfied. However, any combination of unsatisfied clauses will result in ϕ being false. To ensure reversibility, verification of the overall formula must be completed systematically. A diagram illustrating the flow control of the procedure is given in Fig. 10b. The verification is initiated by consuming the signal $\phi^?$ and will complete either by producing the signal ϕ^T, if ϕ is satisfied, or by producing the signal ϕ^F, otherwise. Note that the signals for the variable assignment (which are manipulated by the counter) only act as catalysts in this procedure. Each clause is checked in order. Suppose the formula is not satisfied. Then the first clause i to be unsatisfied will produce the signal C_i^F. This is followed by a reaction that not only produces the ϕ^F signal, but also produces a unique history signal which records the first unsatisfied clause. This history signal ensures that the entire process remains logically reversible. Should the overall formula be satisfied, the ϕ^T signal will be produced. Once ϕ^F or ϕ^T is produced, the verification is complete.

5 Unique Challenges in Natural Computing

Consider the additional challenges of computing with a *soup* of interacting molecules, such as DNA. At once the molecules form the hardware and the software of the system. The state of the computation is denoted only by the presence or absence of certain signals, and this in turn fully dictates which reactions can next occur. Despite this, constructions such as the stack machine and the BRGC counter demonstrate that not only is logically reversible computation possible in these systems, it can also be Turing-universal and space-efficient. However, both constructions share a common assumption: certain signal molecules in the initial set must occur as a single copy. Theoretical results have emerged that suggest these systems will not operate correctly when this single copy assumption is violated [4,6]. To illustrate this point, we again consider the 3-bit BRGC counter. Initially, in a single copy of the construction, the signal molecules $\{0_3, 0_2, 0_1\}$ denote the state $0_3 0_2 0_1$. Consider a two-copy network where the initial multiset of present signal molecules is duplicated, yielding the multiset $\{0_3, 0_3, 0_2, 0_2, 0_1, 0_1\}$. (We also assume a duplicate multiset of transformers is available.) As in the single copy case, assume reaction (1) occurs in the forward direction, followed by reaction (2) in the forward direction. The resulting multiset of signal molecules is $\{0_3, 0_3, 0_2, 1_2, 0_1, 1_1\}$. In the single copy case, we intend that reaction (1) in the reverse direction will occur next; however, given the current multiset of present signal molecules in the two-copy case, reaction (3) in the forward direction could instead occur, resulting in the multiset $\{0_3, 1_3, 0_2, 1_2, 0_1, 1_1\}$. At this point, a copy of every signal molecule is present, and any reaction can occur, in either direction. Furthermore, the single copy case required *at least* seven reactions to produce the final state $1_3 0_2 0_1$, whereas the two-copy case can reach it in three. Crosstalk between the copies has broken the counter. The intuition as to why the single copy assumption is important is that it gives us a means to (temporarily) *erase* information. In a single copy setting, once a molecule of a particular type is consumed, it is no longer present. In a multi-copy setting, once a molecule of a particular type is consumed, there is no guarantee that the other copies are simultaneously consumed.

While the single copy restriction permitted us to study systems pushing the limits of computation for a *chemical soup*, it imposes a significant engineering challenge. All DSD implementations to-date use concentrations of strands of each type. Producing and successfully executing a DSD with a single copy restriction is currently challenging, but feasible. For instance, the first published result on the measurement of a single enzyme molecule was by Boris Rotman, in 1961 [21]. The experimental techniques developed in that first paper are still influential and in use, and new advancements in single molecule studies continue to be made [10].

6 Conclusion

Progress in natural computing, both on the theoretical and experimental side, has been steady. New, experimentally viable computing models, such as DNA strand displacement systems, have inspired promising results to realize logically and physical reversible computing systems using nucleic acids. In this brief tutorial, we have highlighted some of these results and discussed their underlying ideas. These constructions we studied are theoretical, have not been experimentally realized, and are not without unique, practical challenges. An underlying assumption in all constructions surveyed is that certain signal species occur within the reaction volume as a single copy. This contrasts sharply with DNA based computations that have been experimentally realized — copies of each type of strand typically number in the millions or billions. While it may be possible to ensure the single-copy assumption is met, another promising direction is to tether strands to a surface [17,11], such as a DNA origami tile [19].

Finally, we find the current complexity classes for logically reversible computation too general to capture the realities of logically reversible chemical reaction networks (CRNs) and DNA strand displacement systems (DSDs). The class ReversibleSPACE represents all problems that can be solved by a space-bounded logically reversible Turing machine. As with any Turing machine, the space bound is with respect to the length of tape necessary to complete the computation. In CRNs and DSDs, bits of information are represented with the presence and absence of signal molecules. Thus, the length of tape required in the Turing machine computation corresponds well with the maximum quantity of signals required during the CRN computation. However, this does not account for *fuel* (transformers) that a CRN may require to complete its computation. The reaction is the fundamental operation in a CRN just as a state transition is the fundamental operation for a Turing machine. However, with current technology, a reaction in a CRN requires *fuel*, which in turn requires physical space, whereas a Turing machine state transition does not. In essence, a logically reversible Turing machine could perform all state transitions in only one direction, while still using significantly less space than the number of computation steps. This is not currently possible in molecular programming. It has been demonstrated that any space-bounded computation can be realized with a logically reversible CRN that requires only one copy of fuel species (transformers) per reaction equation [25]. It is conceivable this CRN could be simulated with a logically reversible Turing machine. It is also conceivable that such a simulation could be constructed to ensure that each state transition of the Turing machine either strictly alternates being applied in the forward and reverse direction, or adheres to a polynomial bound in the difference between forward and reverse transitions, at every step of the computation. Such a construction would be a logically reversible Turing machine (with symmetric transitions), capable of simulating any space-bounded Turing computation, that is semantically restricted to capture the notion of *fuel*. We let ReversibleSPACE$^\bigstar$ denote the class of problems solvable by such a Turing machine. It has already been shown by Lange *et al.* [13] that ReversibleSPACE = SPACE. An open question is whether ReversibleSPACE$^\bigstar$ = SPACE.

Acknowledgments. The author would like to thank the reviewers for their suggestions.

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science 266(5187), 1021–1024 (1994)
2. Bennett, C.H.: Logical reversibility of computation. IBM Journal of Research and Development 17(6), 525–532 (1973)
3. Cardelli, L.: Two-domain DNA strand displacement. Developments in Computational Models 26, 47–61 (2010)
4. Chen, H.-L., Doty, D., Soloveichik, D.: Deterministic function computation with chemical reaction networks. In: Stefanovic, D., Turberfield, A. (eds.) DNA 2012. LNCS, vol. 7433, pp. 25–42. Springer, Heidelberg (2012)
5. Chiniforooshan, E., Doty, D., Kari, L., Seki, S.: Scalable, time-responsive, digital, energy-efficient molecular circuits using DNA strand displacement. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 25–36. Springer, Heidelberg (2011)
6. Condon, A., Hu, A.J., Maňuch, J., Thachuk, C.: Less haste, less waste: on recycling and its limits in strand displacement systems. Journal of the Royal Society: Interface Focus 2(4), 512–521 (2012)
7. Edmonds, J.: Paths, trees, and flowers. Canadian Journal of mathematics 17(3), 449–467 (1965)
8. Fredkin, E., Toffoli, T.: Conservative logic. International Journal of Theoretical Physics 21, 219–253 (1982)
9. Genot, A.J., Bath, J., Turberfield, A.J.: Reversible logic circuits made of DNA. Journal of the American Chemical Society 133(50), 20080–20083 (2011)
10. Knight, A.E.: Single enzyme studies: A historical perspective. In: Mashanov, G.I., Batters, C. (eds.) Single Molecule Enzymology. Methods in Molecular Biology, vol. 778, pp. 1–9. Humana Press (2011)
11. Lakin, M.R., Phillips, A.: Modelling, simulating and verifying turing-powerful strand displacement systems. In: Cardelli, L., Shih, W. (eds.) DNA 17 2011. LNCS, vol. 6937, pp. 130–144. Springer, Heidelberg (2011)
12. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development 5(3), 183–191 (1961)
13. Lange, K.J., McKenzie, P., Tapp, A.: Reversible space equals deterministic space. Journal of Computer and System Sciences 60(2), 354–367 (2000)
14. Lecerf, Y.: Machines de Turing réversibles. Récursive insolubilité en $n \in N$ de l'équation $u = \Theta^n$, où Θ est un "isomorphisme de codes". Comptes Rendus 257, 2597–2600 (1963)

15. Lewis, H.R., Papadimitriou, C.H.: Symmetric space-bounded computation. Theoretical Computer Science 19(2), 161–187 (1982)
16. Qian, L., Soloveichik, D., Winfree, E.: Efficient Turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011)
17. Qian, L., Winfree, E.: Scaling up digital circuit computation with dna strand displacement cascades. Science 332(6034), 1196–1201 (2011)
18. Qian, L., Winfree, E., Bruck, J.: Neural network computation with DNA strand displacement cascades. Nature 475(7356), 368–372 (2011)
19. Rothemund, P.W.K.: Folding DNA to create nanoscale shapes and patterns. Nature 440(7082), 297–302 (2006)
20. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology 2(12), e424 (2004)
21. Rotman, B.: Measurement of activity of single molecules of β-D-galactosidase. Proceedings of the National Academy of Sciences of the United States of America 47(12), 1981 (1961)
22. Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N.V., Goodman, M.F., Rothemund, P.W.K., Adleman, L.M.: A sticker-based model for DNA computation. Journal of Computational Biology 5(4), 615–629 (1998)
23. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science 314(5805), 1585–1588 (2006)
24. Seeman, N.C.: Nucleic acid junctions and lattices. Journal of Theoretical Biology 99(2), 237–247 (1982)
25. Thachuk, C.: Space and energy efficient molecular programming and space efficient text indexing methods for sequence alignment. PhD thesis, University of British Columbia (2012)
26. Thachuk, C., Condon, A.: Space and energy efficient computation with DNA strand displacement systems. In: Stefanovic, D., Turberfield, A. (eds.) DNA 2012. LNCS, vol. 7433, pp. 135–149. Springer, Heidelberg (2012)
27. Winfree, E.: Algorithmic self-assembly of DNA. PhD thesis, California Institute of Technology (1998)
28. Yurke, B., Turberfield, A.J., Mills, A.P., Simmel, F.C., Neumann, J.L.: A DNA-fuelled molecular machine made of DNA. Nature 406(6796), 605–608 (2000)