Reversible Computation Using Swap Reactions on a Surface

Tatiana Brailovskaya, Gokul Gowri(✉), Sean Yu, and Erik Winfree

California Institute of Technology, Pasadena, CA 91125, USA
{tbrailov,ggowri,ssyu,winfree}@caltech.edu

Abstract. Chemical reaction networks (CRNs) and DNA strand displacement systems have shown potential for implementing logically and physically reversible computation. It has been shown that CRNs on a surface allow highly scalable and parallelizable computation. In this paper, we demonstrate that simple rearrangement reactions on a surface, which we refer to as swaps, are capable of physically reversible Boolean computation. We present designs for elementary logic gates, a method for constructing arbitrary feedforward digital circuits, and a proof of their correctness.

1 Introduction

In traditional digital logic, information is lost, making computation logically irreversible. For example, an AND gate transforms two inputs into a single output, where the inputs cannot be deduced from the output. Landauer argued that irreversible logic, implemented by physically irreversible systems, dissipate at least a minimum energy, $kT \log 2$, with each binary computational step [1]. Charles Bennett countered that computation could be done in a logically reversible fashion, indicating that physically reversible systems could compute with arbitrarily little energy expenditure [2]. A surprising flip side to this discovery was that several simple models of constant-energy reversible systems, such as perfect billiard balls [3,4] and even a microscopic model of heat diffusion within molecular aggregation [5], were shown to be capable of carrying out arbitrary computations. The connections between computation and thermodynamics are now richly developed [6,7]. However, despite considerable effort [8], practical computing systems that perform reversible computing with asymptotically minimal energy expenditure have not been demonstrated.

In his seminal work, Bennett references biological nucleic acid systems as examples of logically and physically reversible computing [2,9]. Recently, building on techniques for compiling arbitrary formal chemical reaction networks (CRNs) into DNA strand displacement systems (DSDs) [10–12], the potential of synthetic nucleic acid systems to implement reversible computing has been explored theoretically and shown to be feasible for polynomial-space problems [13–15]. By further storing information in a DNA polymer based system,

T. Brailovskaya, G. Gowri and S. Yu—Equal contribution.

C. Thachuk and Y. Liu (Eds.): DNA 25, LNCS 11648, pp. 174–196, 2019.
https://doi.org/10.1007/978-3-030-26807-7_10

a scheme for physically and logically reversible Turing-universal computing has been proposed [16]. However, neither of these approaches allow for parallel computing, and they require many distinct species.

This motivated a framework for computing using chemical reaction networks on a surface (surface CRNs) [17]. In a surface CRN, a bimolecular reaction $A + B \rightarrow C + D$ can occur if a molecule of species A is adjacent to a molecule of species B on the surface. A C molecule will replace the A molecule, and a D molecule will replace the B molecule. Note that the molecules can be adjacent in any orientation, such that $A + B \rightarrow C + D \equiv B + A \rightarrow D + C$; both are distinct from $A + B \rightarrow D + C$. The surface CRN framework can be used to construct massively parallelizable space-bounded Turing machines and continuously active logic circuits of different sizes with a constant set of species [17].

In the proposed DNA implementation of surface CRNs [17], species are bound to a DNA origami surface, and free-floating fuel molecules are consumed (and waste molecules produced) to facilitate irreversible reactions between two neighboring species via DNA strand displacement [17]. While one could simulate a reversible surface CRN by utilizing pairs of irreversible reactions of the form $\{A + B \rightarrow C + D; C + D \rightarrow A + B\}$, it is plausible that a genuinely reversible implementation could be devised such that the waste of the forward reaction is the fuel of the reverse reaction, and vice versa (as is the case for some DSD implementations of well-mixed solution CRNs [16]). Implementing such a surface CRN in DNA would involve using one fuel molecule for the forward reaction, and a different fuel molecule for the reverse reaction. In a closed system, where there is no external power maintaining fuels and wastes at constant concentrations, an occurrence of the forward reaction would bias the system toward the reverse direction, as the amount of fuel for the forward direction decreases, while the amount of fuel for the reverse reaction increases. Computation using these pseudo-reversible reactions would be difficult to drive forward unless the system guarantees that each reaction is used equally in each direction, on average [13–15]. Such restrictions impose difficult design constraints.

Another approach to constructing reversible surface CRNs is to use only reactions of the form $A + B \rightarrow B + A$, which are implicitly reversible because surface CRNs do not consider absolute orientation. When the forward reaction's waste is the reverse reaction's fuel, there is therefore no net change in fuels or wastes – the implementation is effectively catalytic. These reactions are simply the rearrangement of two neighboring molecules, henceforth referred to as swaps, and abbreviated as $A \leftrightarrow B$.

Swap reactions were previously discussed as a way to simulate diffusion on a surface [17], but we have found that they can exhibit much more complex programmable behavior. One may initially think that systems built using only swap reactions cannot perform useful computation. No new species can be introduced after computation begins, so NOT gates and signal fanout may at first seem infeasible. Furthermore, since reactions are completely reversible, it may also seem that computation cannot be biased to proceed forward. Indeed, if swap

reactions were implemented in a well-mixed solution rather than on a surface, they would be utterly useless.

However, because we are considering swap reactions on a surface, we can take advantage of the geometry of the surface and local nature of the swaps. We are able to obtain behavior that is far more controlled than random diffusion. In this paper, we will show that by carefully designing initial configurations and swap reactions, arbitrary digital logic circuits can be computed using reversible reactions on a surface.

2 Computing Paradigm

We seek to construct a set of species, along with a set of permissible swap rules and a starting arrangement of those species on the lattice, that is able to compute logical functions.

To represent bits, we will use two species, denoted 1 and 0. Each lattice point will be shown as a square. Within an arrangement of species on a surface, certain lattice points will be designated as input locations and certain lattice points will be designated as output locations. We will denote the input locations with arrows pointing towards them and outputs with arrows pointing away (e.g. see the arrows in the leftmost and rightmost locations in Fig. 1a). In our constructions, if the input lattice locations are replaced with any permutation of bits, then it is possible to reach an arrangement (through the permissible swaps) where the output locations are filled with bits. We can think of this abstractly as the arrangement computing some Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}^m$, where n, m are the number of input and output lattice locations respectively, so long as a unique set of output values are reachable.

Consider the task of constructing a wire. Let's define a species w that is permitted to swap both with 0 and 1 (i.e. we permit the reactions $0 \leftrightarrow W$ and $1 \leftrightarrow W$). Imagine a line of W, as shown in Fig. 1.

If we let the leftmost lattice point be the input location and rightmost be the output location, then this arrangement computes the identity function. The input bit will randomly walk along the line of W, eventually occupying the output location. Any line of W is a wire, as it simply transmits a signal. In order to develop more complex systems, we will introduce more species and rules, and use larger layouts. With this paradigm in mind, we design composable logic gates, which allow us to construct arbitrary digital logic circuits.

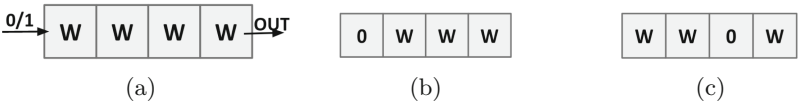


Fig. 1. Basic wire operation. **(a)** Wire starting configuration. **(b)** Initial loaded configuration. **(c)** Computation in progress.

3 Building Logic Circuits with Swaps

We have designed a constant set of 16 species and 23 swap rules, shown in Table 1, that are capable of universal Boolean logic, through the composition of NOT, AND, OR, fanout, and wirecross gate layouts on a surface.

Table 1. List of swap rules required to implement NOT, AND, OR, fanout and wirecross. No species beyond those that appear in these swap rules are necessary. Recall that the rule $A \leftrightarrow B$ is equivalent to the implicitly reversible surface CRN reaction $A + B \rightarrow B + A$

1. $0 \leftrightarrow W$	5. $0 \leftrightarrow I$	9. $I0 \leftrightarrow J$	13. $I0 \leftrightarrow X$	17. $I0 \leftrightarrow J0$
2. $1 \leftrightarrow W$	6. $1 \leftrightarrow I$	10. $I1 \leftrightarrow J$	14. $I1 \leftrightarrow X$	18. $I1 \leftrightarrow J1$
3. $0 \leftrightarrow W0$	7. $0 \leftrightarrow I0$	11. $W0 \leftrightarrow P$	15. $W0 \leftrightarrow X$	19. $W0 \leftrightarrow P0$
4. $1 \leftrightarrow W1$	8. $1 \leftrightarrow I1$	12. $W1 \leftrightarrow P$	16. $W1 \leftrightarrow X$	20. $W1 \leftrightarrow P1$
	21. $I \leftrightarrow J$	22. $K \leftrightarrow 1$	23. $K \leftrightarrow I$	

3.1 NOT, AND, OR

For these gates, we had several important goals in mind, the most important being that the gates had to be logically reversible. Fredkin and Toffoli demonstrate in [3] a set of universal logic gates that are reversible called the Fredkin and the Toffoli gates. Each of them is a three input - three output gate in which it is uniquely possible to determine the input from the output.

In the case of AND, OR logic, this is impossible (since the gate is not injective). However for our systems, it is sufficient to be able to uniquely determine the input to a gate from the output *plus* the final configuration of the gate. The idea is that even though different inputs give the same output, the way in which inputs rearrange in the gate will be unique.

NOT Logic. Initially, it may seem impossible to construct a NOT gate using only swap reactions, as the output of a NOT gate is not present in the input, and swap reactions cannot generate new species. However, we are able to implement a NOT gate by instead storing both a 1 and a 0 inside the gate, and releasing the appropriate bit depending on the input.

The starting configuration of a NOT gate is shown in Fig. 2a. In this configuration, no swaps are possible. Upon the arrival of a bit in the input location, it goes either right or left depending on its value, as 1 swaps with $W1$ and not $W0$, and 0 swaps with $W0$ and not $W1$. Upon this first transition, the bit may now swap with the I species, which may now travel down the two consecutive J species, and release the appropriate output bit. The output bit can then swap into the output location. This motif is symmetric for both inputs, as shown in Fig. 2b. An example trajectory for the 0 input case is shown in Fig. 2c.

Statespace. In order to give a more quantitative explanation of what is happening in this swap system, we have to introduce the notion of the state space as a graph. The nodes of said graph are unique configurations on the 2D surface. Two nodes are connected by an edge if a single swap can take one vertex to the other. From this point forward this graph will be referred to as the statespace graph. For instance, the state space graph for the NOT gate can be seen in Fig. 2d. It is simply a linear graph of 7 nodes, which corresponds to the sequence of states shown in Fig. 2c.

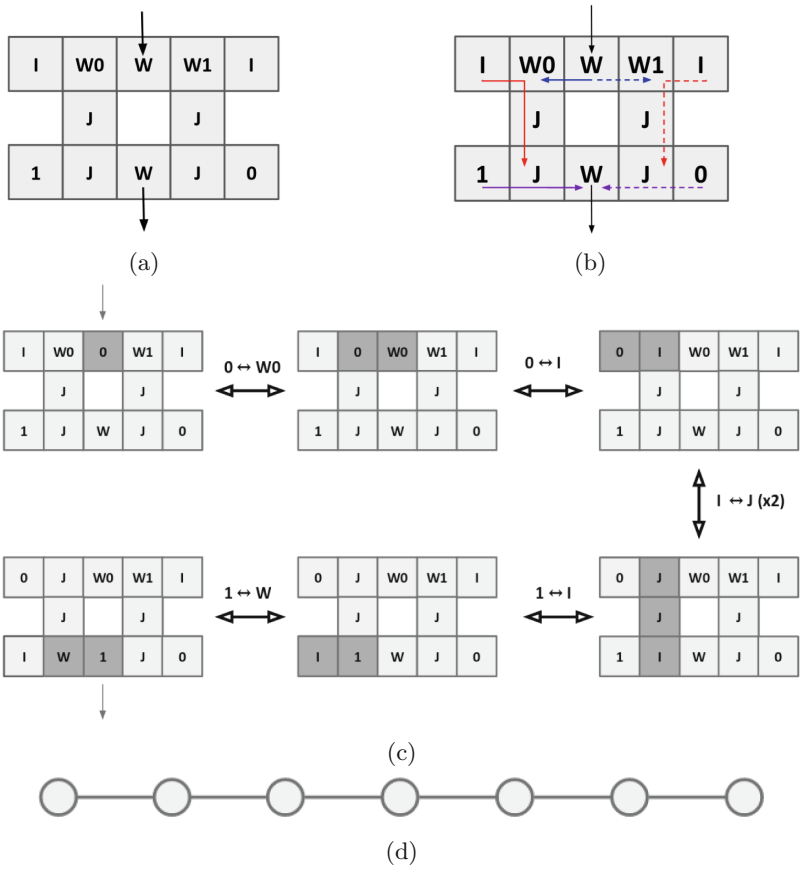


Fig. 2. NOT gate implementation via swap reactions. The unidirectional black arrows indicate the input and output locations. **(a)** Gate layout and swap rules. **(b)** Condensed representation of computation. Blue lines represent the initial movement of the input bit. The red represents the trajectory of the I species. The purple line is the movement of the output bit, which occurs after interacting with the I species. Dotted lines refer to when the input is 1, while solid lines are for input 0. **(c)** Computation trajectory on input 0. The bidirectional black arrows represent transition via swap rule. **(d)** Statespace of the NOT gate. (Color figure online)

All of our logic gates (AND, OR, NOT) have the property that there is only one state in which a correct output is in the output square. This means that in whatever circuit we embed our gates into, when the output leaves the gate, the remainder of the gate can only be in one possible configuration. In particular with the logic gates that we have constructed, after the output bit leaves, the remainder of the gate is stable (where stable means that no swaps can happen between the species in the remaining configuration). Furthermore, the gates are stable before any inputs arrive. This stability property is achieved without sacrificing reversibility by leveraging spatial separation.

AND Logic. Next, we will show how a small change in the NOT gate construction yields an AND gate. Consider a gate with the same layout as NOT except that the 1 and 0 species that are stored inside the gate are switched. If this gate receives an input 1, the output will be 1. Similarly, if the gate receives input 0, the output with be 0. Now, further modify this gate by removing the stored 1 and turning it into another input location, as shown in Fig. 3a. Observe that an AND gate will return 0 whenever there is a 0 in the input. Thus, in this construction, if the top input receives a 0, the gate will correctly output 0, regardless of what value arrives at the side input. It remains to verify that this gate works correctly when the top input receives a 1. Whenever the top input receives a 1, the side input will appear in the output location. Thus, if the side input receives 0, the output will be 0, as it should. Similarly, if the side input and top input both receive 1, the output will be 1, as desired.

OR Logic. Even though AND and NOT are sufficient for expressing any logical formula, it is convenient to also have a simple way of computing OR directly. Observe that in an OR gate, whenever there is a 1 in the input, the output should be 1. Thus, we can create OR analogously to AND, by fixing 1 as a stored input that always appears if the top input is 1. The layout is shown in Fig. 3b.

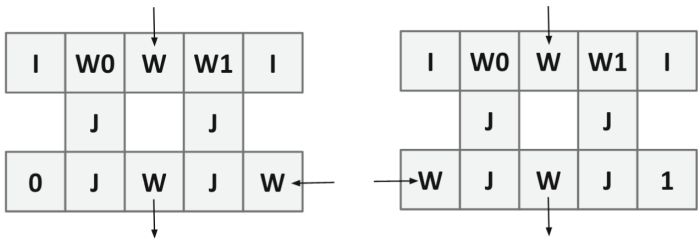


Fig. 3. Gate layout of AND and OR. For these two gates we have two inputs, denoted by the arrows pointing into the system. **(a)** AND gate. **(b)** OR gate.

Selector Motif. If no bits are stored in the previously described gates such that they have three input locations (top, left, and right), then the gate is equivalent to a data selector (or multiplexer), a device frequently used by electrical engineers that outputs one of two values determined by an additional input signal. We have shown the implementation of AND, OR, and NOT by fixing certain inputs, but we can also implement several other functions with this strategy, including strictly less than and strictly greater than. However, in our constructions, we use only the canonical AND, OR, and NOT gates.

The selector design was originally inspired by the Fredkin gate [3]. The Fredkin gate works by switching signals two and three in the output if signal one is 1, else the signals are output unchanged. A Fredkin gate can be programmed by fixing one or two of the three input signals and observing the output value of either signal two or three to compute AND, OR and NOT. Here, we effectively construct a Fredkin gate that only outputs the relevant signal that stores the value of AND, OR, or NOT.

3.2 Fanout, Wirecross

Although they are not essential for universality, wirecross and fanout are very helpful in creating compact circuits. It has been shown that it is possible to construct a wirecross from just fanout and elementary logic gates [18]. Wirecross seems challenging on a two dimensional surface, as wires cannot use the third

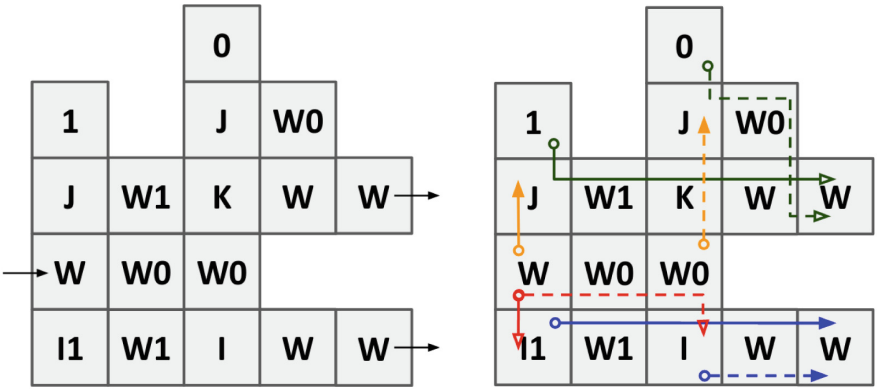


Fig. 4. Fanout mechanism. **(a)** Fanout layout. The input arrives on the left side indicated by the arrow. The two outputs leave on the right at the specified locations. **(b)** Fanout computation. The dotted lines represent the computation when a 0 bit serves as the input and the solid lines represent computation when a 1 bit serves as the input. The red lines represent the trajectory of the input. The yellow lines are the trajectory of the $I1$ and I species for the cases in which the input is 1 and 0 respectively. The green lines are the movement of the top output bit and the blue lines are the movement of the bottom output bit. (Color figure online)

dimension to avoid intersection. Fanout also seems challenging since we cannot create new species via swaps. We are able to implement wirecross by effectively constructing two unique wire types, and we are able to implement fanout by storing the additional bits inside the gate.

Fanout. In Fig. 4a we demonstrate our design for a fanout gate. The key idea is to store an additional copy of the bit inside the gate to simulate bit duplication. Overall, the gate consists of two interconnected columns, each of which is capable of copying a bit. One column will duplicate the input bit if it is 1 and the other will duplicate the input if it is 0. The bit is "duplicated" by freeing the extra stored bit of the same value.

If the input bit is a 1, then the first column duplicates it, and the two outputs travel on wires parallel to the $W0$ wire. We introduce a new species K that can both carry the 1 bit and the indicator I species. We also have to introduce a new species $I1$ to allow the 1 bit to duplicate. Refer to Table 1 for the associated swap rules.

Wirecross. Designing a wirecross that allows two bits to cross paths is challenging because if the signals are travelling along identical wire types, there is nothing stopping them from trading places and going down the wrong wires. In order to get around this issue, we translate the bits into intermediates that travel on distinct wire types and then release the appropriate bit once the crossing has taken place. We see this design in Fig. 5. At the top, we translate the input bit $a \in \{0, 1\}$ to $Wa \in \{W0, W1\}$. This Wa then travels along a path specified by the P species. The other input b is translated to $Ib \in \{I0, I1\}$, which travels along a path specified by the J species. The center species is initially occupied

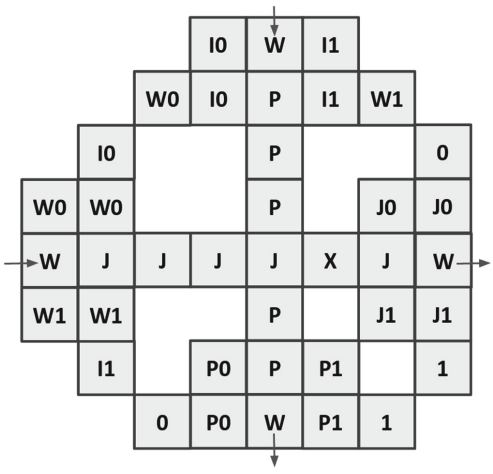


Fig. 5. Wirecross layout.

with a J, which means the input coming in from the left can pass, while the input from the top must wait. We introduce a final new species X which can swap with both Ib and Wa. First the Ib traverses the intersection and places the X species in the center. Then the displaced Wa can traverse the intersection point.

3.3 Compiling a Feedforward Circuit to a Surface Layout

With our gate constructions, we can compile arbitrary feedforward logic circuits (involving only AND, OR, NOT, fanout and wirecross) into surface layouts that evaluate the same functions via swap reactions. Here we describe one method of compilation of a circuit to a layout, following a standard crossbar array architecture [19]. This is not a spatially efficient way of constructing a surface layout; it is merely here to demonstrate that any feedforward circuit can be laid out on a surface.

We define a feedforward circuit as a graph by associating a circuit with a directed graph in the following way (for an explicit example of such a graph see Fig. 6a). Let the nodes denote inputs, outputs and gates (AND, OR, NOT). The in-edge at node b from node a indicates that output of node a is fed as input to the gate at node b. Similarly, an out-edge from node a to node b indicates that output of a is fed into the gate at node b. If such a graph is acyclic, then we say that the corresponding circuit is feedforward.

For simplicity, assume that all gate, fanout and wirecross layouts fit within a square of dimensions $s \times s$ lattice points, with sufficient space to route inputs on the top and left to outputs on the bottom and right. Our construction places the inputs regularly along the top of a rectangular region, and places the gates regularly along an offset diagonal, with inputs and gates each providing their output along vertical wires directly below them, extending to the bottom of the rectangle. To route the appropriate inputs to each logic gate, we use horizontal wires extending left from the gate, using the wirecross to pass through undesired wires and using the fanout to acquire the desired input signal while leaving that signal available for downstream gates that might also want it. The feedforward order of the circuit ensures that we can order the gates along the diagonal such that the inputs for a gate are always available to its left. As shown in Fig. 6b, this leads to a complete circuit layout of size not much larger than $(n + m)s \times 2\,ms$ lattice points for a circuit with n input and m gates, which is worst-case asymptotically optimal [20].

3.4 4-Bit Square Root Circuit

Following the examples in refs. [21] and [17], we designed a system that computes the floor of the square root of a 4-bit binary number. The layout, shown in Fig. 7, was not created using the procedure described in Sect. 3.3, but rather was designed by hand to be more compact. The correctness of the circuit was verified through exhaustive enumeration of the state space for every input, with long wires abbreviated in order to reduce simulation time. Since the system is

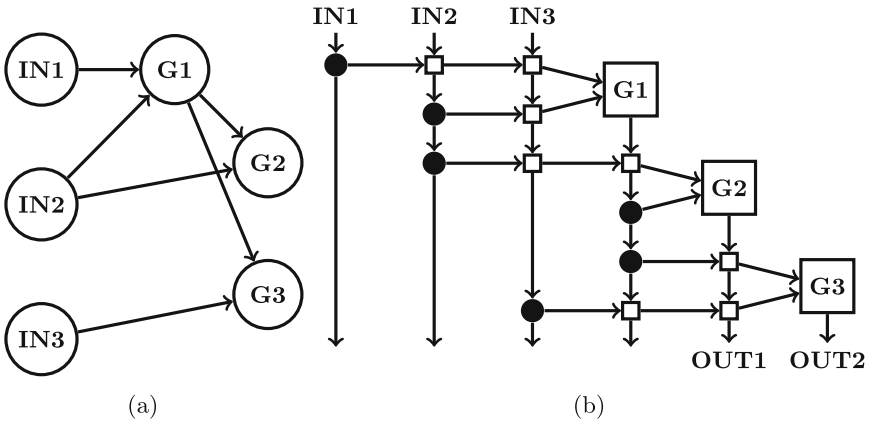


Fig. 6. A systematic feedforward circuit layout compilation. The circuit analyzed here computes $G2(G1(IN1, IN2), IN2)$ and $G3(G1(IN1, IN2), IN3)$. $G1$, $G2$, $G3$ are gates, $IN1$, $IN2$, $IN3$ are inputs, and $OUT1$, $OUT2$ are outputs. **(a)** Directed acyclic graph specification of the desired circuit. **(b)** Surface layout of the given circuit generated according to procedure described in Sect. 3.3. Filled circles indicate fanout. Empty squares indicate wirecross.

stochastic, parallel, and reversible, correctness was evaluated not with respect what the system *does* do, but rather with respect to what it *could* do, i.e. which states are reachable from a given input state. There was no combination of inputs for which the exhaustive enumeration found a state in which any incorrect output was produced, and for all input combinations the enumeration found a state in which all correct outputs were produced. Therefore, for all input combinations, the circuit will (with probability 1) eventually reach a state where all outputs are correct, and will never reach a state where any output is incorrect (though perhaps most of the time the outputs will be empty).

4 Proving Circuit Correctness

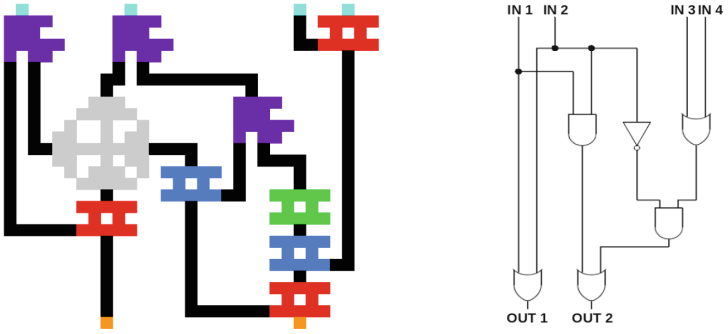
Leaving aside the question of kinetics for now, we will proceed to argue for the general correctness of feedforward circuit layouts, in the above sense. Our argument relies on establishing the composability of our gates in a surface CRN using strong and weak bisimulation similar to as in [22,23]. We start by showing that each gate is equivalent to a simple stochastic CRN. Then, we show that a composition of the stochastic CRNs for individual gates is equivalent to a composition of gates in a surface CRN. Finally, we will show that the composed stochastic CRN is logically equivalent to the intended function. In other words, when gates are composed on the surface CRN, the function they compute is exactly the logical composition of the gates.

When constructing an equivalent stochastic CRN for each gate in the surface CRN, we require that, upon starting with the set of species corresponding to

bits being loaded into the input locations in the gate, the set of states of the stochastic CRN will be in bisimulation with the configuration of the gate on the surface when the same input bits are loaded. Observe that each gate's operation involves a set of independent random walks that come together at particular points. For example if we consider the OR gate, the top input takes a random walk and ends up in the top right corner. Meanwhile the indicator species I takes a random walk from its original location in the top right to the bottom right. If we give each of these random walks a label, then the entire state of the gate can be described by how far each random walk trajectory has progressed and which bit is being expressed on the trajectory. The species of our stochastic CRN will have the form A_i^b where A is the label of the trajectory, i the index of the trajectory (how far it has progressed) and b is the bit that is being expressed on the trajectory. We also allow i to be S or F indicating respectively that the trajectory has not started yet (it is in the starting configuration) or that the trajectory is finished.

The most basic reaction of the stochastic CRN is $A_i^b \leftrightarrow A_{i+1}^b$ indicating progress on trajectory A (provided A_{i+1}^b is an allowed species). When we have one trajectory lead unconditionally into another we see a reaction that looks like $A_n^b + B_S \leftrightarrow A_F^b + B_1^b$. When two trajectories come together in the surface CRN layout, this corresponds to reactions consuming the final numbered species of two trajectories and producing the first numbered species of a new trajectory.

In order for the stochastic CRN to be equivalent we must show that there is a bisimulation between the state space of the surface CRN (given a specific combination of inputs in the input locations) and the set of reachable states in the stochastic CRN (from the corresponding set of starting species). To demonstrate



(a) Surface Layout (b) Circuit Diagram

Fig. 7. 4-bit square root circuit. **(a)** Layout of the surface CRN 4-bit square root (done by hand in order to use less space than the general circuit layout scheme). Locations marked in purple represent the fanout mechanism. Red represents an OR gate, blue represents an AND, green a NOT, and grey a wirecross. Lattice points in light blue are input locations for the circuit and orange represents output locations. Black represents wires. **(b)** Abstract circuit diagram. (Color figure online)

this we will have a paint function f that maps states of the stochastic CRN to configurations (states) of the surface gate. This mapping will have the property that the available swaps from a state $f(s)$ on the surface correspond exactly to the set of reactions available to the stochastic CRN state s. By this we mean that applying the paint function f and then applying the swap will result in the same thing as applying the CRN reaction corresponding to the swap, and then applying the paint function. The paint function operates as follows, each species will specify how to paint a part of the surface. The parts of the surface that are not specified by any species will be the same as its initial configuration. We demonstrate a paint function for the OR gate in Fig. 8.

	A	B	C	D	E
1	I	W1	W	W0	I
2		J		J	
3	1	J	W	J	W
4			W		

(a)

Species	C1	D1	E1	D2	D3
A_S	W				
$A_1^{(0)}$	0	W0	I	J	J
$A_2^{(0)}$	W0	0	I	J	J
$A_3^{(0)}$	W0	I	0	J	J
$A_4^{(0)}$	W0	J	0	I	J
$A_5^{(0)}$	W0	J	0	J	I
$A_F^{(0)}$	W0	J	0	J	

(b)

Species	C3	C4
D_S	W	W
$D_1^{(0)}$	0	W
$D_2^{(0)}$	W	0
$D_F^{(0)}$	W	W
$D_1^{(1)}$	1	W
$D_2^{(1)}$	W	1
$D_F^{(1)}$	W	W

(c)

Species	E3
B_S	W
$B_1^{(0)}$	0
$B_F^{(0)}$	I
$B_1^{(1)}$	1
$B_F^{(1)}$	I

(d)

Species	D3
C_S	
$C_1^{(0)}$	0
$C_F^{(0)}$	W
$C_1^{(1)}$	1
$C_F^{(1)}$	W

(e)

Species	C1	B1	A1	B2	B3	A3
$A_1^{(1)}$	1	W1	I	J	J	1
$A_2^{(1)}$	W1	1	I	J	J	1
$A_3^{(1)}$	W1	I	1	J	J	1
$A_4^{(1)}$	W1	J	1	I	J	1
$A_5^{(1)}$	W1	J	1	J	I	1
$A_6^{(1)}$	W1	J	1	J	1	I
$A_F^{(1)}$	W1	J	1	J	W	I

(f)

Fig. 8. Paint functions for the OR gate. All species in the stochastic CRN that correspond to the OR gate as well as their associated paint functions are shown. In each table, the left hand column is the list of species. Each row is a paint function for a particular species. Each column represents how a particular lattice point is painted. For example, the $A_F^{(0)}$ species would paint the lattice point C1 as $W0$, the lattice point D1 as J, and so on. **(a)** Initial configuration of the gate with each lattice point labeled with a row number and a column letter. **(b)** Paint function for $A^{(0)}$ including A_S. **(c)** Paint function for D. **(d)** Paint function for B. **(e)** Paint function for C. **(f)** Paint function for $A^{(1)}$ excluding A_S.

The reactions for the stochastic CRN that correspond to the OR gate are

$$\begin{aligned} A_5^{(0)} + B_1^{(0)} + C_S &\leftrightarrow A_F^{(0)} + B_F^{(0)} + C_1^{(0)} \\ A_5^{(0)} + B_1^{(1)} + C_S &\leftrightarrow A_F^{(0)} + B_F^{(1)} + C_1^{(1)} \\ C_1^{(0)} + D_S &\leftrightarrow C_F^{(0)} + D_1^{(0)} \\ C_1^{(1)} + D_S &\leftrightarrow C_F^{(1)} + D_1^{(1)} \\ A_6^{(1)} + D_S &\leftrightarrow A_F^{(1)} + D_1^{(1)} \\ X_i^{(b)} &\leftrightarrow X_{i+1}^{(b)} \quad \forall X \in \{A, D\}, \forall b \in \{0, 1\}, \forall i \end{aligned} \tag{1}$$

If we load the inputs b_1, b_2 on the top and on the right of the OR gate respectively, the equivalent state in the stochastic CRN is to have one copy each of $A_1^{(b_1)}$ and $B_1^{(b_2)}$ along with C_S and D_S. One can verify by hand that, starting from these initial four species, reachable reactions in the stochastic CRN correspond exactly to reachable swaps on the surface. In fact whenever at most one input arrives at either input location, the stochastic CRN is in (strong) bisimulation with the surface CRN. If for example b_1 entered through the top input, but nothing had entered through the right input, we can start the system in the state $\{A_1^{(b_1)}, B_S, C_S, D_S\}$. We represent this CRN pictorially in Fig. 9a.

Now notice that if we drop all subscripts in our species formulation, then the stochastic CRN reduces to

$$\begin{aligned} A^{(0)} + B^{(0)} + C &\leftrightarrow A^{(0)} + B^{(0)} + C^{(0)} \\ A^{(0)} + B^{(1)} + C &\leftrightarrow A^{(0)} + B^{(1)} + C^{(1)} \\ C^{(0)} + D &\leftrightarrow C^{(0)} + D^{(0)} \\ C^{(1)} + D &\leftrightarrow C^{(1)} + D^{(1)} \\ A^{(1)} + D &\leftrightarrow A^{(1)} + D^{(1)} \end{aligned} \tag{2}$$

This simplification of the CRN has a nice interpretation. If we think of each trajectory label A as some wire in part of the system, the species $A^{(b)}$ has the interpretation of "wire A is hot and is carrying bit b". Now having an input of b_1, b_2 like described above is equivalent to starting with $\{A^{(b_1)}, B^{(b_2)}, C, D\}$. This reduced stochastic CRN is in weak bisimulation with the full stochastic CRN, for valid initial conditions. In Fig. 9b, we show a further simplified version of this CRN. The nodes in this graph represent distinct species (written out explicitly in Fig. 9a). Edges in this graph represent reactions with one node adjacent to that edge being a product and the other a reactant that additionally involves species C or D. Thus, Fig. 9b shows how different combinations of inputs $A^{(b_1)}$, $B^{(b_2)}$ are combined to obtain $D^{OR(b_1,b_2)}$. Thus this reduced stochastic CRN makes it easy to see that our gate works as expected, with A, B being the input trajectories and D being the output trajectory.

The equivalent paint functions for the stochastic CRNs for AND and NOT are straightforward extensions of the above. The equivalent paint function for

the stochastic CRNs for them and for fanout and wirecross can be found in the Appendix. What happens now when we have a composition of gates in which the output lattice points of certain gates are linked with input lattice points of other gates via a (possibly bent) linear wire? We can create a "stitched" version of the stochastic CRN. We start with the stochastic CRN for each gate, and we relabel the species so that the same trajectories from two different OR gates, for example, are syntactically different. Next, we consider each wire in the composition. Suppose a given wire W links the output trajectory O of some gate to the input trajectory I of some other gate. Then we create a new trajectory for the movement of bits on a wire—let us call it W—and add the following reactions:

$$\begin{aligned} O_n^{(b)} + W_S &\leftrightarrow O_F^{(b)} + W_1^{(b)} \\ W_l^{(b)} + I_S &\leftrightarrow W_F^{(b)} + I_1^{(b)} \\ W_i^{(b)} &\leftrightarrow W_{i+1}^{(b)} \quad \forall b \in \{0,1\}, \forall i \text{ s.t. } 1 \leq i < l \end{aligned} \tag{3}$$

where l is the length of the wire and $O_n^{(b)}$ is the last state of the output trajectory. In the reduced CRN this is effectively $O^{(b)} \leftrightarrow I^{(b)}$, which makes sense. Since we have already established every reaction in each gate's individual

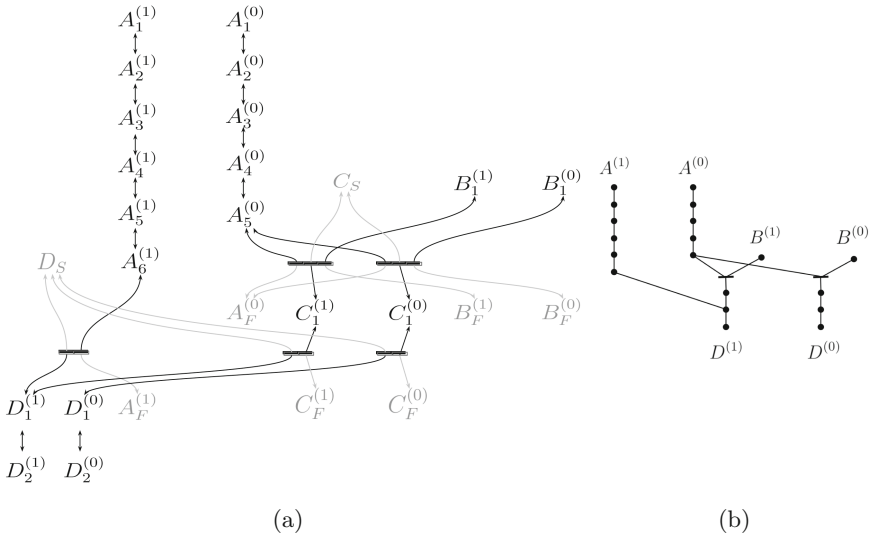


Fig. 9. Pictorial representation of the stochastic CRN for the OR gate.**(a)** The full stochastic CRN. Species and arrows shown in gray correspond to species that serve as indicators of whether a certain trajectory is in an initial or a final configuration. **(b)** A graph representation of the CRN without the grey indicator species. All species that only differ in their subscript are associated with each other. Note that the node above $D^{(1)}$ that appears to have two possible predecessors is actually deterministically reversible when the indicator species are accounted for.

stochastic CRN corresponds to some swap on the surface, its fairly clear that within this stitched stochastic CRN, all reactions still correspond to reachable swaps on the surface. What is left to show is that no reachable swaps cannot be expressed as some reaction in the stitched stochastic reaction. First, we claim that if the circuit is feedforward and each of its input locations is loaded with exactly one bit, then the circuit as a whole receives exactly 1 input per input wire. Consider what happens if this is not the case. Suppose a gate receives two inputs at an input lattice. Consider the first such occurrence. Let g be the first gate this happened to. g must have received two inputs from a previous gate, some g' that it was connected to. But g' could not have produced two outputs since we know g' received at most one input per lattice (by our selection of g). Therefore we have a contradiction, and so no such g can exist. If each gate will receive at most one input per input wire, the behavior of each gate is entirely captured by its individual stochastic CRN. Because the stochastic CRN for the circuit is a stitching of individual stochastic CRNs it must capture all possible swaps. Therefore the stitched stochastic CRN is in bisimulation with the surface CRN.

5 Kinetics and Entropics

As discussed above, for each gate there is only one state with correct output. Therefore, in a circuit in which every wire leads toward an output lattice point (such as the one in Fig. 7a), there is also only one state with all outputs simultaneously occupied. This is problematic for large circuits because the probability of observing an output state becomes extremely small as the circuit increases in size. The source of the issue is that forward and reverse reaction rates are identical for a single swap reaction. Thus, every possible configuration of the surface has the same energy. As a result, at equilibrium each reachable surface configuration is equally likely. In the case of the 4-bit square root, there is exactly one state in which we have outputs, and $> 10^6$ states in which we do not. As such, probability that the surface is in the output state is very low.

Conveniently, the reduced stochastic CRN provides a framework for quantitatively assessing entropic factors, which in turn suggests a solution to the problem. Consider a "wire" of length n that is represented in the full stochastic CRN by $\{X_S, X_1, \ldots, X_n, X_F\}$ and is represented in the reduced stochastic CRN by $\{X_S, X, X_F\}$. We say that a signal is on the wire if some X_i with $1 \leq i \leq n$ is present in the full stochastic CRN, and if X is present in the reduced stochastic CRN. A state α of the reduced stochastic CRN with signals on m wires $X^1, X^2, \ldots, X^m$ (of respective lengths $n_1, n_2, \ldots, n_m$) will therefore correspond to exactly $w_\alpha = \prod_{1 \leq k \leq m} n_k$ states of the full stochastic CRN. Consequently, we consider α to be a macrostate with Boltzmann weight w_α and thus equilibrium probability $p_\alpha = w_\alpha / Z$ where $Z = \sum_\alpha w_\alpha$. Equivalently, we could say that macrostate α has energy $E_\alpha = -kT \sum_{1 \leq k \leq m} \log n_k$.

As an initial application of this quantitative framework, we can determine the effect of the "unnecessary" fanout gates in the construction of Fig. 6b. These

gates were introduced in the construction as a convenience so that input signals and computed signals can be propagated down vertical wires, just in case they will be needed later on – and we did not bother to complicate the construction by specifying how to remove the unnecessary wires and fanout gates. Yet they have an interesting side effect: system states with signals on all the output locations will also have signals trapped on the unnecessary wires. Thus, rather than there being exactly one such output state, there will be many—for example, the product of the lengths of the unnecessary wires, in a simple case. These wires therefore, by their presence, bias the computation forward. By introducing additional unnecessary fanout gates throughout the circuit, we could further bias each step of the computation to ensure that there are no deep valleys in the macrostate energy landscape that could kinetically trap the system. Noting that a fanout gate can reverse itself only if *both* its output signals come back, we can see that it provides a probabilistic kinetic ratchet as well.

As a more compact alternative to this use of fanout gates for their entropic side-effects, we introduce a compact entropic driver mechanism (Fig. 10) that biases computation forward. The mechanism results in an inflation of the number of states after a bit has traversed the mechanism.

Consider loading the gadget in Fig. 10 with a 0 bit species on the left. There is only one state in which there is a 0 bit species on the leftmost lattice point. However, if that bit travels to the right side, then we observe that the number of states in which the 0 bit species is on the rightmost point is 4. This is because after the bit travels to the right side, each of the I and J species and can swap; the entropy gadget has been "activated". Thus each of the pairs can either swap or not, which means we have $2 \times 2 = 4$ possible configurations.

If we have an entropic gadget placed after a gate G_1 connected via a wire to gate G_2, the output bit is less likely to return into G_1 and more likely to enter G_2, than if there was no entropic gadget present. This happens for two reasons. First, the J species could be blocking the wire for the bit to move backwards, favoring progression of computation forward. Second, the presence of the entropic gadget quadruples the number of states in which the bit has entered G_2, thus increasing the probability that the bit is in G_2. Thus, the entropic gadget can be used to increase the likelihood of observing the output state of a circuit.

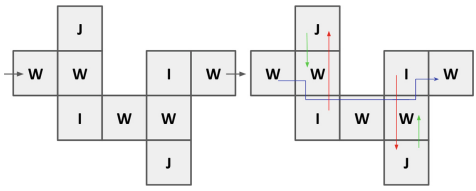


Fig. 10. The entropic driver gadget. The blue line indicates the trajectory of the primary bit traveling down the wire. The red and green lines represent the movement of the I and J species after the primary bit has passed. (Color figure online)

By adjusting the size or the number of entropic driver mechanisms, we can control the increase in number of states, and bias computation forward to an arbitrary degree. For instance, if we change the number of Js from 1 to n by adding more Js on top and bottom, on the left and the right of the entropic gadget, respectively, the number of states of the entropic gadget will be $(n + 1)^2$. With larger n, we further increase the likelihood that computation moves forward. Alternatively, we can put many entropic gadgets next to each other. A wire consisting of n gadgets in series will have 4^n states at the end of the wire. As computation proceeds, more entropy gadgets become activated and the number of possible configurations is exponential in the number of gadgets activated. Thus, by including sufficiently many entropy gadgets or increasing the number of J species, we can drive computation forward to an arbitrary extent.

By judiciously introducing entropy driver gadgets, it should be possible to modify any circuit such that the energies of the macrostates are decreasing roughly linearly as a function of number of gates completed, and thus computation is thermodynamically driven forward. Note that the entropy thus produced by circuit operation entails a corresponding energetic cost to performing the computation.

6 Future Directions

In this work, we constructed arbitrary feedforward circuits using reversible swap reactions on a surface. We also devised a entropic driver mechanism to tunably drive computation forward. However, many fascinating questions regarding swap reaction systems remain unanswered. For instance, the circuits we construct above are not reusable. Might it be possible to devise renewable swap systems that, for example, implement reversible sequential logic or reversible Turing machines using swap reactions?

When we first thought about swap reactions, a primary motivation was that due to the simplicity of such a reaction, there may exist a simple molecular implementation. The DNA strand displacement mechanism that was originally proposed for implementing arbitrary bimolecular reactions on a surface [17] is complex and potentially not suitable for experimental implementation. Now that the computational potential of swap rules has been established, the search for a simpler, more experimentally feasible mechanism is well motivated.

Acknowledgements. Support from National Science Foundation grant CCF-1317694 is gratefully acknowledged. We also thank Lulu Qian and Chris Thachuk for helpful discussion and comments.

Appendix

	A	B	C	D	E
1	I	W1	W	W0	I
2		J		J	
3	0	J	W	J	1
4			W		

(a)

Species	C3	C4
B_S	W	W
$B_1^{(1)}$	1	W
$B_2^{(1)}$	W	1
$B_1^{(0)}$	0	W
$B_2^{(0)}$	W	0

(b)

Species	C1	D1	E1	D2	D3	E3
A_S	W					
$A_1^{(0)}$	0	W0	I	J	J	1
$A_2^{(0)}$	W0	0	I	J	J	1
$A_3^{(0)}$	W0	I	0	J	J	1
$A_4^{(0)}$	W0	J	0	I	J	1
$A_5^{(0)}$	W0	J	0	J	I	1
$A_6^{(0)}$	W0	J	0	J	1	I
$A_F^{(0)}$	W0	J	0	J	W	I

(c)

Species	C1	B1	A1	B2	B3	A3
$A_1^{(1)}$	1	W1	I	J	J	0
$A_2^{(1)}$	W1	1	I	J	J	0
$A_3^{(1)}$	W1	I	1	J	J	0
$A_4^{(1)}$	W1	J	1	I	J	0
$A_5^{(1)}$	W1	J	1	J	I	0
$A_6^{(1)}$	W1	J	1	J	0	I
$A_F^{(1)}$	W1	J	1	J	W	I

(d)

$$A_i^{(b)} \leftrightarrow A_{i+1}^{(b)} \ \forall b \in \{0,1\}$$

$$A_6^{(b)} + B_S \leftrightarrow A_F^{(b)} + B_1^{(\neg b)} \ \forall b \in \{0,1\}$$

(e)

Fig. 11. Paint functions for the NOT gate. All species in the stochastic CRN that corresponds to the NOT gate as well as their associated paint functions are shown. In each table, the left hand column is the list of species. Each row is a paint function for a particular species. Each column represents how a particular lattice is painted. **(a)** Initial configuration of the NOT gate with each lattice point labeled with a row number and a column letter. **(b)** Paint function for B. **(c)** Paint function for $A^{(0)}$ including A_S. **(d)** Paint function for $A^{(1)}$ excluding A_S. **(e)** Stochastic CRN equivalent to the NOT gate.

	A	B	C	D	E
1	I	W1	W	W0	I
2		J		J	
3	W	J	W	J	0
4			W		

(a)

Species	C1	B1	A1	B2	B3
$A_1^{(1)}$	1	W1	I	J	J
$A_2^{(1)}$	W1	1	I	J	J
$A_3^{(1)}$	W1	I	1	J	J
$A_4^{(1)}$	W1	J	1	I	J
$A_5^{(1)}$	W1	J	1	J	I
$A_F^{(1)}$	W1	J	1	J	

(b)

Species	C3	C4
D_S	W	
$D_1^{(0)}$	0	W
$D_2^{(0)}$	W	0
$D_1^{(1)}$	1	W
$D_2^{(1)}$	W	1
D_F	W	W

(c)

Species	A3
B_S	W
$B_1^{(0)}$	0
$B_1^{(1)}$	1
B_F	I

(d)

Species	B3
C_S	
$C_1^{(0)}$	0
$C_1^{(1)}$	1
C_F	W

(e)

Species	C1	D1	E1	D2	D3	E3
A_S	W					
$A_1^{(0)}$	0	W0	I	J	J	0
$A_2^{(0)}$	W0	0	I	J	J	0
$A_3^{(0)}$	W	I	0	J	J	0
$A_4^{(0)}$	W0	J	0	J	I	0
$A_5^{(0)}$	W0	J	0	J	I	0
$A_6^{(0)}$	W0	J	0	J	0	I
$A_F^{(0)}$	W0	J	0	J	W	I

(f)

$$A_6^{(0)} + D_S \leftrightarrow A_F^{(0)} + D_1^{(0)}$$
$$A_5^{(1)} + B_1^{(0)} + C_S \leftrightarrow C_1^{(0)} + A_F^{(1)} + B_F$$
$$A_5^{(1)} + B_1^{(1)} + C_S \leftrightarrow C_1^{(1)} + A_F^{(1)} + B_F$$
$$C_1^{(b)} + D_S \leftrightarrow D_1^{(b)} + C_F \; \forall b \in \{0, 1\}$$
$$X_i^{(b)} \leftrightarrow X_{i+1}^{(b)} \; \forall X \in \{A, D\} \; \forall b \in \{0, 1\}$$

(g)

Fig. 12. Paint functions and the stochastic CRN for the AND gate. **(a)** Initial AND gate surface layout with each lattice labeled with a row number and a column letter. **(b)** Paint function for $A^{(1)}$. **(c)** Paint function for D. **(d)** Paint function for B. **(e)** Paint function for C. **(f)** Paint function for $A^{(0)}$. **(g)** Stochastic CRN equivalent to AND gate.

	A	B	C	D	E
1		IN			
2	I1	W	J	1	
3	W1	W0	W1		
4	I	W0	K	J	0
5	W		W	W0	

(a)

Species	B1	B2	B3	B4
A_S	W	W		
$A_1^{(1)}$	1	W		
$A_2^{(1)}$	W	1		
$A_F^{(1)}$	W			
$A_1^{(0)}$	0	W	W0	W0
$A_2^{(0)}$	W	0	W0	W0
$A_3^{(0)}$	W	W0	0	W0
$A_4^{(0)}$	W	W0	W0	0
$A_F^{(0)}$	W	W0	W0	

(b)

Species	A2	A3	A4	A5
B_S	I1	W1	I	W
$B_1^{(1)}$	1	W1	I	W
$B_2^{(1)}$	W1	1	I	W
$B_3^{(1)}$	W1	I	1	W
$B_4^{(1)}$	W1	I	W	1
$B_F^{(1)}$	W1	I	W	W
$B_1^{(0)}$			0	W
$B_2^{(0)}$			W	0
$B_F^{(0)}$			W	W

(c)

Species	B2	C2	D2	C3	C4	C5
C_S		J	1	W1	K	W
$C_1^{(1)}$	I1	J	1	W1	K	W
$C_2^{(1)}$	J	I1	1	W1	K	W
$C_3^{(1)}$	J	1	I1	W1	K	W
$C_4^{(1)}$	J	W1	I1	1	K	W
$C_5^{(1)}$	J	W1	I1	K	1	W
$C_6^{(1)}$	J	W1	I1	K	W	1
$C_F^{(1)}$	J	W1	I1	K	W	W

(d)

Species	B4	C4	D4	E4	D5	C5
$C_1^{(0)}$	I	K	J	0	W0	W
$C_2^{(0)}$	K	I	J	0	W0	W
$C_3^{(0)}$	K	J	I	0	W0	W
$C_4^{(0)}$	K	J	0	I	W0	W
$C_5^{(0)}$	K	J	W0	I	0	W
$C_6^{(0)}$	K	J	W0	I	W	0
$C_F^{(0)}$	K	J	W0	I	W	W

(e)

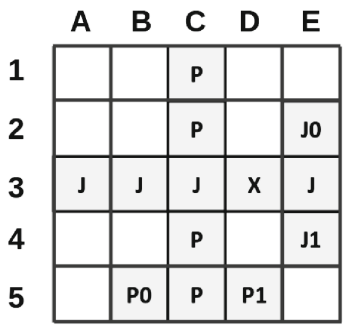
$$A_2^{(1)} + B_S + C_S \leftrightarrow A_F^{(1)} + B_1^{(1)} + C_1^{(1)}$$

$$A_4^{(0)} + B_S + C_S \leftrightarrow A_F^{(0)} + B_1^{(0)} + C_1^{(0)}$$

$$X_i^{(b)} \leftrightarrow X_{i+1}^{(b)} \quad \forall X \in \{A, B, C\}, \forall b \in \{0, 1\}$$

(f)

Fig. 13. Paint functions for the fanout gate. **(a)** Initial configuration of the fanout gate with each lattice labeled with a row number and a column letter. **(b)** Paint function for A. **(c)** Paint function for B. **(d)** Paint function for $C^{(1)}$ including C_S. **(e)** Paint function for $C^{(0)}$ excluding C_S. **(f)** Stochastic CRN equivalent to fanout.



(a)

Species	A3	B3	C3
A_S	J	J	J
$A_1^{(1)}$	I1	J	J
$A_2^{(1)}$	J	I1	J
$A_3^{(1)}$	J	J	I1
$A_F^{(1)}$	J	J	
$A_1^{(0)}$	I0	J	J
$A_2^{(0)}$	J	I0	J
$A_3^{(0)}$	J	J	I0
$A_F^{(0)}$	J	J	

(b)

Species	D3	E3	E2	E4
B_S	X	J	J0	J1
$B_1^{(1)}$	I1	J	J0	J1
$B_2^{(1)}$	J	I1	J0	J1
$B_3^{(1)}$	J	J1	J0	I1
$B_F^{(1)}$	J		J0	I1
$B_1^{(0)}$	I0	J	J0	J1
$B_2^{(0)}$	J	I0	J0	J1
$B_3^{(0)}$	J	J0	I0	J1
$B_F^{(0)}$	J		I0	J1

(c)

Species	C1	C2
C_S	P	P
$C_1^{(1)}$	W1	P
$C_2^{(1)}$	P	W1
$C_F^{(1)}$	P	
$C_1^{(0)}$	W0	P
$C_2^{(0)}$	P	W0
$C_F^{(0)}$	P	

(d)

Species	C3	C4	C5
D_S	X	P	P
$D_1^{(1)}$	W1	P	P
$D_2^{(1)}$	P	W1	P
$D_3^{(1)}$	P	P	W1
$D_F^{(1)}$	P	P	
$D_1^{(0)}$	W0	P	P
$D_2^{(0)}$	P	W0	P
$D_3^{(0)}$	P	P	W0
$D_F^{(0)}$	P	P	

(e)

$$X_i^{(b)} \leftrightarrow X_{i+1}^{(b)} \; \forall X \in \{A, B, C, D\} \, \forall b \in \{0, 1\}$$

$$A_3^{(b)} + B_S \leftrightarrow A_F^{(b)} + B_1^{(b)} + D_S \; \forall b \in \{0, 1\}$$

$$C_2^{(b)} + D_S \leftrightarrow C_F^{(b)} + D_1^{(b)} \; \forall b \in \{0, 1\}$$

(f)

Fig. 14. Paint function for partial wirecross. Truncated portions are equivalent to trajectories found in AND/OR/NOT gates. **(a)** Initial configuration of the central portion of wirecross with each lattice labeled with a row number and a column letter. **(b)** Paint function for A. **(c)** Paint function for B. **(d)** Paint function for C. **(e)** Paint function for D. **(f)** Stochastic CRN equivalent to central portion of wirecross.

References

1. Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. **5**, 183–191 (1961)
2. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**, 525–532 (1973)
3. Fredkin, E., Toffoli, T.: Conservative logic. Int. J. Theor. Phys. **21**, 219–253 (1982)
4. Margolus, N.: Physics-like models of computation. Phys. D Nonlinear Phenom. **10**, 81–95 (1984)
5. D'Souza, R.M., Homsy, G.E., Margolus, N.H.: Simulating digital logic with the reversible aggregation model of crystal growth. In: Griffeath, D., Moore, C. (eds.) New Constructions in Cellular Automata, pp. 211–230. Oxford University Press, Oxford (2003)
6. Ouldridge, T.E.: The importance of thermodynamics for molecular systems, and the importance of molecular systems for thermodynamics. Nat. Comput. **17**, 3–29 (2018)
7. Wolpert, D.: The stochastic thermodynamics of computation. J. Phys. Math. Theor. **52**, 193001 (2019)
8. Perumalla, K.S.: Introduction to Reversible Computing. Chapman and Hall/CRC, Boca Raton (2013)
9. Bennett, C.H.: The thermodynamics of computation-a review. Int. J. Theor. Phys. **21**, 905–940 (1982)
10. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Natl. Acad. Sci. **107**, 5393–5398 (2010)
11. Chen, Y.-J., et al.: Programmable chemical controllers made from DNA. Nat. Nanotechnol. **8**, 755–762 (2013)
12. Srinivas, N., Parkin, J., Seelig, G., Winfree, E., Soloveichik, D.: Enzyme-free nucleic acid dynamical systems. Science **358**, eaal2052 (2017)
13. 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). https://doi.org/10.1007/978-3-642-32208-2_11
14. Codon, A., Hu, A.J., Manuch, J., Thachuk, C.: Less haste, less waste: on recycling and its limits in strand displacement systems. Interface Focus **2**, 512–521 (2012)
15. Condon, A., Thachuk, C.: Towards space- and energy-efficient computations. In: Kempes, C., Grochow, J., Stadler, P., Wolpert, D. (eds.) The Energetics of Computing in Life and Machines, Chap. 9, pp. 209–232. The Sante Fe Institute Press, Sante Fe (2019)
16. Qian, L., Soloveichik, D., Winfree, E.: Efficient turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18305-8_12
17. Qian, L., Winfree, E.: Parallel and scalable computation and spatial dynamics with DNA-based chemical reaction networks on a surface. In: Murata, S., Kobayashi, S. (eds.) DNA 2014. LNCS, vol. 8727, pp. 114–131. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11295-4_8
18. Goldschlager, L.M.: The monotone and planar circuit value problems are log space complete for P. ACM SIGACT News **9**, 25–29 (1977)
19. Masson, G.M., Gingher, G.C., Nakamura, S.: A sampler of circuit switching networks. Computer **12**, 32–48 (1979)

20. Savage, J.E.: Models of Computation. Addison-Wesley, Reading (1998). Section 12.6
21. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science **332**, 1196–1201 (2011)
22. Milner, R.: Communication and Concurrency. Prentice Hall, Upper Saddle River (1989)
23. Johnson, R.F., Dong, Q., Winfree, E.: Verifying chemical reaction network implementations: a bisimulation approach. Theor. Comput. Sci. **765**, 3–46 (2019)