Nat Comput
DOI 10.1007/s11047-015-9503-8

Emulating cellular automata in chemical reaction–diffusion networks

Dominic Scalise[1] · **Rebecca Schulman**[1,2]

Abstract Chemical reactions and diffusion can produce a wide variety of static or transient spatial patterns in the concentrations of chemical species. Little is known, however, about what dynamical patterns of concentrations can be reliably *programmed* into such reaction–diffusion systems. Here we show that given simple, periodic inputs, chemical reactions and diffusion can reliably emulate the dynamics of a deterministic cellular automaton, and can therefore be programmed to produce a wide range of complex, discrete dynamics. We describe a modular reaction–diffusion program that orchestrates each of the fundamental operations of a cellular automaton: storage of cell state, communication between neighboring cells, and calculation of cells' subsequent states. Starting from a pattern that encodes an automaton's initial state, the concentration of a "state" species evolves in space and time according to the automaton's specified rules. To show that the reaction–diffusion program we describe produces the target dynamics, we simulate the reaction–diffusion network for two simple one-dimensional cellular automata using coupled partial differential equations. Reaction–diffusion based cellular automata could potentially be built in vitro using networks of DNA molecules that interact via branch migration processes and could in principle perform universal computation, storing their state as a pattern of molecular concentrations, or deliver spatiotemporal instructions encoded in concentrations to direct the behavior of intelligent materials.

Keywords Reaction–diffusion · Cellular automata · DNA strand displacement · Chemical reaction network · Intelligent materials · Molecular programming · Programmable matter · Distributed computation

1 Introduction

A fundamental question in materials design is how we might program materials to sense and respond to dynamic signals across time and space. Biological materials routinely exhibit this capacity, as cells and tissues sense and respond to a complex array of spatial and temporal cues. For example, during chemotaxis, many cells can detect gradients of chemoattractants and move in the direction of increasing chemoattractant concentration. In a mechanism like chemotaxis (Murray 2003; Greenfield et al. 2009; Baker et al. 2006), cells use spatiotemporal chemical reaction networks to process information collected by distributed chemical sensors to decide on and execute responses to changing environmental conditions. In this paper we discuss the design of analogous synthetic chemical reaction networks that have robust, programmable spatiotemporal dynamics. The ability to engineer such systems could have wide-ranging applications for the design of smart, responsive, programmable materials.

To design a generic set of mechanisms that can process a wide range of input signals and invoke a wide range of responses, we consider a framework for distributed spatial computation that has been studied extensively—the cellular automaton (Fig. 1). A cellular automaton (CA) is a model of computation consisting of a rectangular lattice of

✉ Rebecca Schulman
rschulm3@jhu.edu

Dominic Scalise
scalise@jhu.edu

[1] Department of Chemical and Biomolecular Engineering, Johns Hopkins University, Baltimore, MD, USA

[2] Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA

Published online: 05 June 2015

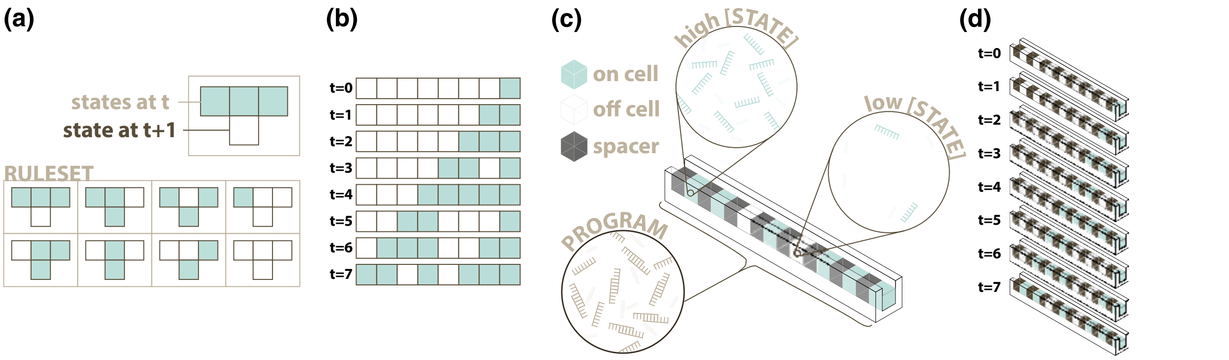

Fig. 1 *A cellular automaton* consists of a lattice of cells. At a given time, each cell is in one of a finite number of states, shown by *color* (*blue* or *white*). Cell states change over time as the result of the application of local rules—finite functions that take as inputs the states of the current cell and a finite set of neighbors and produce the cells' new state as output. Here we consider a one-dimensional automaton where each cell is either *on* or *off*, and where update rules take the cell's own state and those of its left and right neighbors as inputs. **a** An example rule set. **b** Example dynamics for the rule set in (**a**). **c** Schematic of the chemical reaction–diffusion cellular automaton described in this paper. A one-dimensional channel contains cells separated by spacers. The state in each cell is encoded by either a *high* or *low* concentration of a 'state' species within that cell. Spacers between cells, which do not contain any state information, are shown in black. During the computation, the program and state species react and diffuse. This reaction–diffusion process maintains and updates cell state according to the rules of the desired cellular automaton. **d** Target dynamics of the state species for the example cellular automaton rule in (**a**)

domains, or 'cells'. At a given time a cell can be in one of a finite number of states, such as an *on* or *off*. In a synchronous CA, cells update their state once per time step based on their current state and the current states of a finite set of nearby cells. Although each cell update is relatively simple, groups of cells can together perform elaborate spatial computation. CA can execute any computable algorithm, a trait known as universal computation (Gács 2001; Gács and Reif 1988; Cook 2004; Neary and Woods 2006). Specific automata also exist that can programmably construct any structure (Neumann and Burks 1966; Codd 1968), self-replicate (Neumann and Burks 1966), (Codd 1968; Langton 1984), mutate and evolve (Sayama 1999). CA have previously been implemented in DNA tile assemblies (Rothemund et al. 2004; Fujibayashi et al. 2007), and it has further been suggested that it may be possible to implement CA with reactions tethered to the surface of DNA origami (Qian and Winfree 2014). In the case of DNA tile assembly, cell states are stored in molecular building blocks (i.e. tiles) whose preference for stably binding to other tiles with matching two-input binding sites facilitates each step of the cellular automaton's computation, forming a growing array that stores the entire history of the computation.

In this paper we propose a strategy for building synchronous CA using chemical reaction–diffusion networks. We begin by breaking down CA into their fundamental operations: storage of cell states, communication between nearby cells, and calculation of new cell states. We demonstrate how existing chemical computing mechanisms could implement these operations. We then combine these chemical mechanisms to emulate two specific automata, known as 'Rule 110' and 'Rule 60'. These chemical CA can be viewed as a proof of concept that synthetic materials could sense signals across space and time and execute a broad class of dynamic programmed responses.

2 Background: reaction–diffusion processes for computation

Reaction–diffusion (RD) networks are sets of chemically reactive species that diffuse within a continuous substrate. In contrast to a well-mixed chemical reaction system, reaction–diffusion (RD) networks can produce spatial patterns, where some species are more abundant in some parts of the substrate and less abundant in others. The interplay of reactions and diffusion can lead to sustained spatial patterns and even the emergence of patterns from a homogeneous initial substrate which experiences transient concentration fluctuations (Turing 1952). Transient waves within reaction–diffusion patterns can also generate Turing patterns or perform basic computations (Tóth and Showalter 1995; Steinbock et al. 1996; Bánsági et al. 2011).

Recently it has been shown that arbitrary chemical reaction networks can be readily designed and implemented in vitro using short strands of synthetic DNA (Soloveichik et al. 2010; Chen et al. 2013). Because DNA binding under many conditions is largely determined by the Watson–Crick sequence complementarity (A-T, G-C), reactive species can be designed to minimize unintended crosstalk between species that should not interact. These techniques separate the design of new reaction networks from the discovery of naturally occurring chemicals that perform the intended reactions. In support of this idea, large reaction networks involving up to 130 different sequences of DNA have been demonstrated in vitro without substantial crosstalk (Qian and Winfree 2011), and have been used to implement Boolean logic (Qian and Winfree 2011; Seelig et al. 2006; Qian and Winfree 2011). Further, the rates of the emulated reactions can be controlled (Chen et al. 2013).

It also appears plausible to extend this mechanism of DNA reaction design to the design of large reaction–diffusion networks, as the diffusion rates of different DNA strands can be programmed. Because the diffusion coefficient of DNA scales polynomially with the length of the strand (Smith et al. 1996), the diffusion rate of each species in a DNA reaction network can be independently tuned by adding or removing bases from a sequence, and such changes can be done so that the reactive propensity of a species is largely unchanged. Further, within a polymer gel, species attached to the gel substrate do not diffuse, but can continue to react. Together, the capacities to design arbitrary chemical reactions and tune the diffusion coefficient of each species in principle enable us to implement de novo simple RD networks that perform pattern transformations (Allen et al. 2012; Chirieleison et al. 2013; Dalchau et al. 2014). Here we ask how we might design an RD network that could be implemented by DNA molecules, given what is known about designing DNA-based reactions and diffusion processes. To focus on this question, here we ignore experimental nonidealities and the challenges of building large molecular networks, including unintended crosstalk between species.

By designing RD network modules that perform simple, predictable, repeatable transformations to a pattern of chemical concentrations, circuits of modules can be combined to perform elaborate patterning operations (Scalise and Schulman 2014). Pattern transformation modules take specific species as inputs, perform reactions potentially involving some intermediate species within the module, and produce an output species (Fig. 2). Modules can be connected together with the output of upstream modules serving as the input to downstream modules. If these modules are designed such that the intermediate species of one module do not react with the intermediates of other modules, then many modules can operate simultaneously in the same substrate without interfering with each other. Further, by imposing the design requirement that modules must not significantly deplete (or "load") the concentrations of their inputs, it is possible to ensure that a module's reactions affect only other modules that lie downstream within the network (Scalise and Schulman 2014). Thus, modules can be added one at a time to a system such that each addition of a module results in a simple, predictable change to the patterning process. In the case of modular DNA strand-displacement systems, formal analysis can help to verify the system will operate correctly (Codon et al. 2012; Lakin et al. 2013).

Here we extend existing pattern transformation techniques to emulate a discrete, synchronous, one-dimensional CA, generating spatial patterns of chemical concentrations with controlled dynamics. We design a network of reaction–diffusion pattern transformation modules (defined in detail in Fig. 2) in combination with a simple, static initial pattern and an external "clock" signal whose concentrations change periodically. This network forms the target CA structure, and controllably transforms the state of that structure over time. In principle, our abstract chemical reaction–diffusion network could be translated into a DNA strand displacement network for in vitro implementation.

One challenge in the design of pattern transformations is that the second law of thermodynamics implies that without the continual input of energy, purely diffusive patterns are unstable and tend to become well mixed over time. Thus, to prevent spatial patterns of soluble molecules from dissipating, reaction–diffusion networks will require a constant energy flux. One way to achieve this flux is to develop reactions that slowly release and degrade high-energy species. These reactions produce a sustained flux of molecules in the environment, and maintain a pattern such that only sporadic replenishment of some high-energy precursors are required to sustain the pattern formation process. *Production* reactions take the form *source* → *species*, and continuously produce reactants that are depleted by converting a high-energy precursor into the desired species. Degradation reactions take the form *species* → *waste*, and convert species that are produced into low-energy waste to stabilize the system. We assume that the resulting waste products are inert. In practice, the eventual accumulation of waste products may necessitate their physical extraction from the system.

3 Anatomy of our reaction–diffusion cellular automaton

Reaction–diffusion systems emulating a one-dimensional CA must be able to store the current state of the system as a tape or grid of cells and execute the update rules as a function of the states of the cell and the cell's left and right neighbors (Fig. 3). For the class of CA we consider here, the state of each cell is either *on* or *off*.

In our construction, each cell is a region of the substrate with a static, uniformly high concentration of a particular catalyst molecule. Catalyst molecules are attached to the substrate, so they do not diffuse. We call these molecules

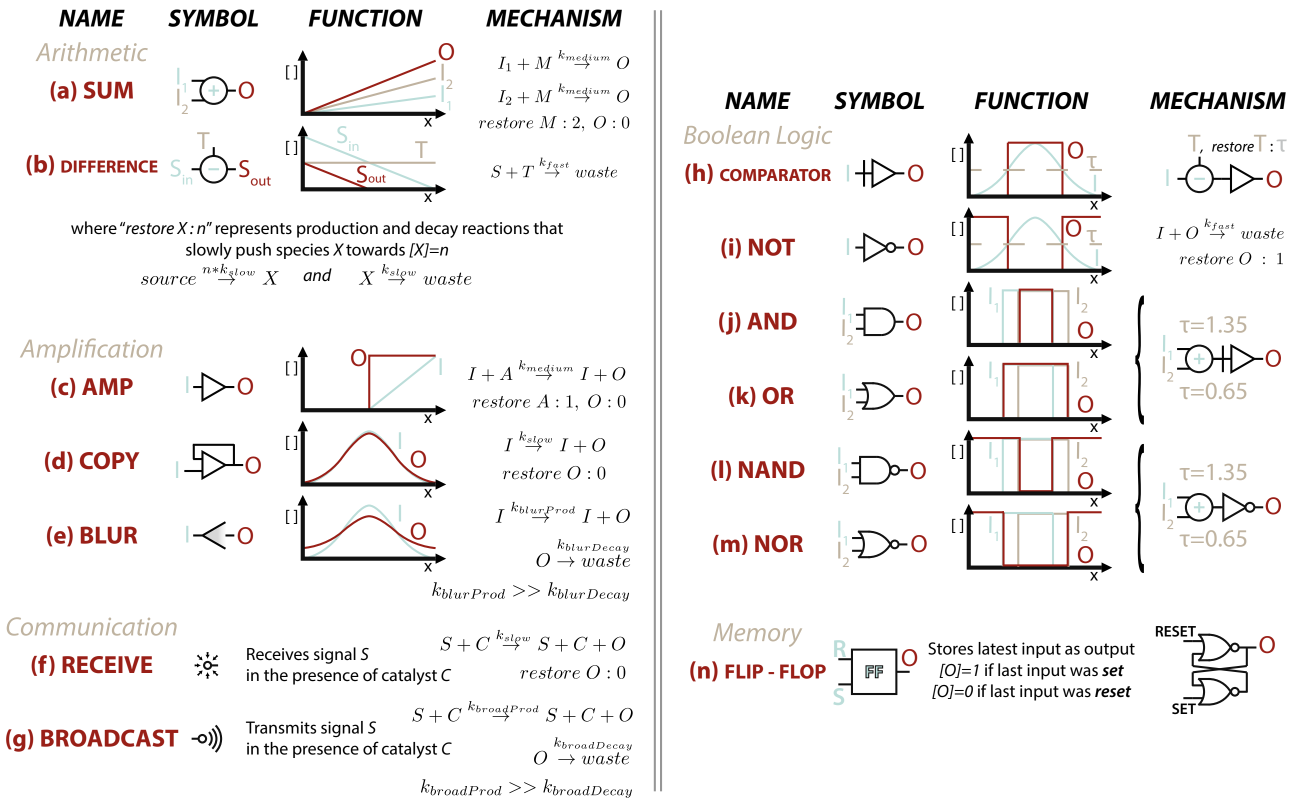
NAME
SYMBOL
FUNCTION
MECHANISM
Arithmetic
(a) SUM
$I_1 + M \xrightarrow{k_{medium}} O$
$I_2 + M \xrightarrow{k_{medium}} O$
$restore\ M : 2,\ O : 0$
(b) DIFFERENCE
$S + T \xrightarrow{k_{fast}} waste$
where "restore X : n" represents production and decay reactions that slowly push species X towards [X]=n
$source \xrightarrow{n*k_{slow}} X$ and $X \xrightarrow{k_{slow}} waste$
Amplification
(c) AMP
$I + A \xrightarrow{k_{medium}} I + O$
$restore\ A : 1,\ O : 0$
(d) COPY
$I \xrightarrow{k_{slow}} I + O$
$restore\ O : 0$
(e) BLUR
$I \xrightarrow{k_{blurProd}} I + O$
$O \xrightarrow{k_{blurDecay}} waste$
$k_{blurProd} >> k_{blurDecay}$
Communication
(f) RECEIVE
Receives signal S in the presence of catalyst C
$S + C \xrightarrow{k_{slow}} S + C + O$
$restore\ O : 0$
(g) BROADCAST
Transmits signal S in the presence of catalyst C
$S + C \xrightarrow{k_{broadProd}} S + C + O$
$O \xrightarrow{k_{broadDecay}} waste$
$k_{broadProd} >> k_{broadDecay}$
Boolean Logic
(h) COMPARATOR
T, restore T : τ
(i) NOT
$I + O \xrightarrow{k_{fast}} waste$
$restore\ O\ :\ 1$
(j) AND
(k) OR
τ=1.35
τ=0.65
(l) NAND
(m) NOR
τ=1.35
τ=0.65
Memory
(n) FLIP - FLOP
Stores latest input as output
[O]=1 if last input was set
[O]=0 if last input was reset
RESET
SET

Fig. 2 *Reaction–diffusion modules* Each module is a set of simpler modules or of abstract chemical reactions that could in principle be emulated in vitro using a DNA strand-displacement network. In an emulation of these reactions using DNA strand displacement processes, species are represented as DNA strands or hybridized complexes of strands with particular sequences. Other strands or complexes are also required for the emulation, and act as "intermediates" in the reaction process (Soloveichik et al. 2010; Scalise and Schulman 2014). More details on these modules and their operation can be found in Scalise and Schulman (2014)

'keys.' In our one-dimensional grid, cells can have one of four different types of keys (Key_A, Key_B, Key_C and Key_D) with key types repeating as one proceeds down the channel so that cells can identify neighbors based on local key type (Fig. 3a). For instance, cells defined by Key_A are always neighbored on the right by Key_B cells, and the left by Key_D cells. In principle it is possible to accomplish the same task using only three keys. We chose to include a fourth key type because with only three types, cells that are two units apart might mistake each other for immediate neighbors if their broadcasts reach a slightly wider area than expected. With four key types, broadcasts would have to span three units in order to cause an error, rather than just two. Cells are separated by 'spacer' regions that do not contain keys.

Since key molecules only participate in reactions as catalysts, it is assumed that they are not depleted over time.[1] In this paper we assume that this pattern of key catalysts is present at the start of the reaction as a set of initial conditions. Such a grid pattern could either be manually patterned into the substrate by top-down techniques, such as microcontact printing (Ruiza and Chen 2007) or directed hydrogel assembly (Du et al. 2008), or generated by a separate bottom-up RD patterning program (Scalise and Schulman 2014).

In addition to the static pattern of key catalysts, a mix of many other freely diffusing "program" species is supplied across the substrate. These program molecules interact with the key molecules to emulate the dynamics of a CA in a cycle of three discrete conceptual stages (Fig. 3b, c). In the first stage, cells share their states with their neighbors and receive the states of other neighbors. Next, cells use information about their local state and the states of their neighbors to determine their state at the next time step. Finally, the calculated state is stored as the current state. Cycles of communication, calculation of new states, and storage of the new state in a cell's memory emulate the dynamics of a CA. In the construction below, we assume the existence of a global chemical "clock" signal which everywhere in space oscillates with a well-defined period and amplitude. Such a clock would be easiest to provide using an external system, but might also be possible using

[1] In practice, catalysts have a fixed turnover number. Reactions that cyclically produce and degrade catalysts could enable the periodic replacement of key catalysts (and other catalysts in the system) that are no longer functional.

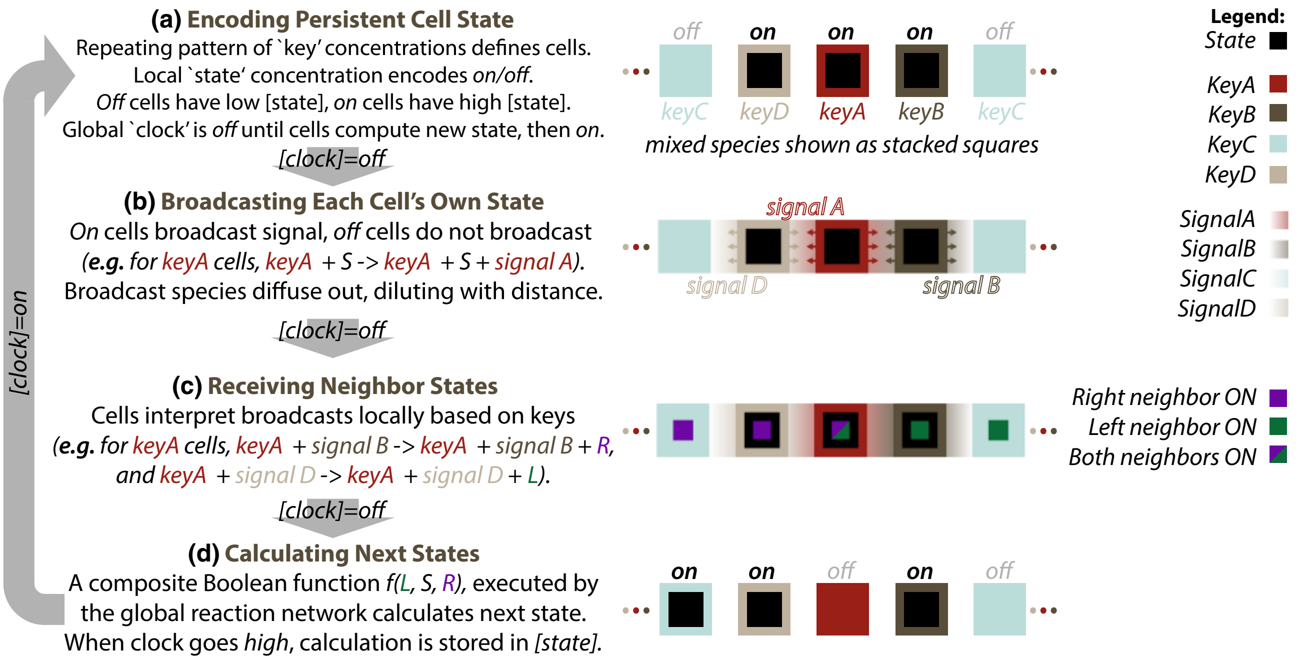
(a) Encoding Persistent Cell State
Repeating pattern of `key' concentrations defines cells.
Local `state' concentration encodes on/off.
Off cells have low [state], on cells have high [state].
Global `clock' is off until cells compute new state, then on.
[clock]=off
(b) Broadcasting Each Cell's Own State
On cells broadcast signal, off cells do not broadcast
(e.g. for keyA cells, keyA + S -> keyA + S + signal A).
Broadcast species diffuse out, diluting with distance.
[clock]=off
(c) Receiving Neighbor States
Cells interpret broadcasts locally based on keys
(e.g. for keyA cells, keyA + signal B -> keyA + signal B + R,
and keyA + signal D -> keyA + signal D + L).
[clock]=off
(d) Calculating Next States
A composite Boolean function f(L, S, R), executed by
the global reaction network calculates next state.
When clock goes high, calculation is stored in [state].
[clock]=on
off
on
on
on
off
keyC
keyD
keyA
keyB
keyC
mixed species shown as stacked squares
signal A
signal D
signal B
on
on
off
on
off
Legend:
State ■
KeyA ■
KeyB ■
KeyC ■
KeyD ■
SignalA ■
SignalB ■
SignalC ■
SignalD ■
Right neighbor ON ■
Left neighbor ON ■
Both neighbors ON ■

Fig. 3 *The chemical CA we describe* performs three types of operations. *a* The state of each cell is stored locally. *b* and *c* Cells communicate their state to their immediate neighbor cells. *d* A Boolean logic circuit calculates each cell's next state as a function of the cell's own state and the state of its neighbors, and stores this new state in (*a*), completing one cycle of automaton dynamics. A global clock signal synchronizes these three operations. The clock is *off* for the communication and calculation stages, and turns *on* to allow the calculated new state to be stored in memory for the next cycle

an oscillatory chemical reaction combined with a system that would maintain synchronization across space (Danino et al. 2010). We included this clock because while asynchronous CA can be constructed without the use of a global clocking mechanism (Nehaniv 2004), our attempts to design unclocked reaction–diffusion systems that emulate CA resulted in dramatically more complex circuits compared to our clocked system. We therefore begin by assuming a clock exists in the system, and in Sect. 5 discuss how we could dispense with this requirement.

3.1 Communication: sending and receiving addressable signals

We begin the description of a CA cycle in the Communication Stage, right after a set of cell states for the last time step have been stably stored. At this point, each cell's current *on*/*off* state is represented, respectively, by the local *high* or *low* concentration of a 'state' species. During the communication stage of a computation cycle, cells must communicate their current state to their immediate neighbors.

Communication is managed by Broadcast and Receive modules (Fig. 4). Each *on* cell broadcasts information about its current state by producing a signal within the cell that can diffuse away from the cell. Broadcast modules

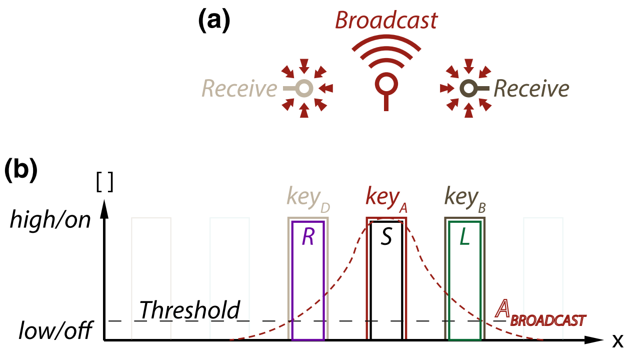

Fig. 4 *Communication stage* **a** Using a broadcast module, each *on* cell produces a gradient encoding its current state and key. Receive modules interpret broadcasts based on the identity of their local key. **b** [*species*] *versus x* for a single *on A*-type cell, with local $[S] = high$. A broadcast module generates a stable gradient of $A_{Broadcast}$ that decays with distance from Key_A. Receive modules at *D*-type cells interpret $A_{Broadcast}$ as *R*, while receive modules at *B*-type cells interpret the same broadcast as *L*. Broadcast signals below a threshold are ignored, so cells only communicate with their neighbors

(Fig. 2g) execute this function. In order for neighboring cells to interpret these broadcasts as coming from the left or right neighbor cell, these broadcasts must contain information about which cell is sending them. The identity of the cell broadcasting information about its state is encoded using the key types of cells: Cells that are defined by 'key

A' species broadcast 'signal A', those defined by 'key B' broadcast 'signal B', and so on. The distance that broadcast signals propagate is controlled by the production and degradation rates of the Broadcast module, such that a cell's broadcasts only reach its neighbors. Each key has its own separate broadcast module.

The counterpart to a Broadcast module, a Receive module (Fig. 2f), receives signals transmitted by a cell's neighbors and translates them into local information about the state of a neighboring cell. This conversion is also done in a cell-specific manner, such that each cell converts particular broadcasts into information about particular types of neighbors. For example, within cells defined by Key_A, the key species catalyzes the conversion of the broadcast signal from Key_B into a species that encodes the *right* neighbor's state as *on* and catalyzes the conversion of the broadcast signal from Key_D into a species that encodes the *left* neighbor's state as *on*. Key species *B* through *D* catalyze a corresponding set of reactions to produce signals that encode whether their *right* and *left* neighbors are *on*. Each conversion reaction of a broadcast to a type of neighbor information is managed by a separate receive module. Because there are four keys and each cell has two neighbors, eight Receive modules are required.

Receive modules convert broadcasts into "preliminary neighbor signals". These preliminary neighbor signals are at different concentrations throughout a cell because they are copies of the broadcast signals, which decay in concentration with the distance from the neighbor. To produce uniform *on*/*off* neighbor signals throughout a cell, comparators (Fig. 2h) rectify preliminary neighbor signals, producing digital "processed neighbor signals" whose *on* levels are the same across a cell. Together, the broadcast and receive modules ensure that after some period of broadcasting, each cell contains species that encode its own state and those of its neighbors.

3.2 Calculation stage: calculating new state changes

Neighbor broadcasts that are received and processed by each cell are used to calculate the next cell state. Each update rule can be encoded as a Boolean circuit with the neighbors and the cell's own state as inputs. Such circuits can be implemented as a set of reaction–diffusion program modules (Fig. 5). For instance, in a Rule 60 CA, a cell's next generation state is *on* if its own current state is *on* OR its left-hand neighbor is *on*, but NOT if both of these states are *on*. Because the state of the right-hand neighbor is irrelevant, Rule 60 cells do not need to listen to their right-hand neighbor's broadcast. The logic for a Rule 110 local update is performed by the sub-circuit in Fig. 5d. The output signal produced by this circuit determines the target state of the cell at the next time step.

3.3 Memory: storing and stably encoding a new cell state

During the Memory stage the computed next state of the cell is stored using a "flip-flop" module (Fig. 6). Flip-flops have "set" and a "reset" input signal and one output signal. When the set input is *on*, the output signal also turns *on*. The output remains *on* even after the set signal turns *off*. When the reset signal turns *on* the output turns *off*, and remains *off* until the set signal again turns *on*. This module encodes the cell's state, providing a persistent state signal used by the communication and calculation stages.

The reactions that communicate a cell's state to its neighbors and compute its next state occur without control over timing. Different cells (or different regions within a cell) may take different amounts of time to compute their new state. To ensure that all cells finish computing their next states before any other cell commits its new state to memory, calculated next states are not stored in memory

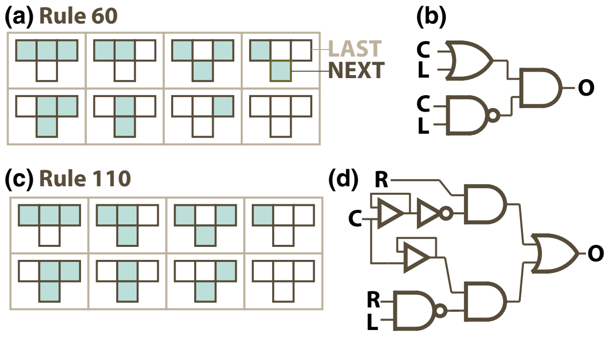

Fig. 5 *Calculation stage* Every CA update rule has a corresponding Boolean logic implementation. **a** Rule 60. **b** Rule 60 converted into Boolean logic. **c** Rule 110. **d** Boolean logic for Rule 110

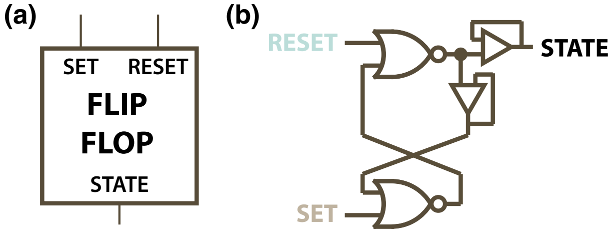

Fig. 6 *The memory stage* Stores cell state in a flip-flop. **a** A flip-flop's output does not change when its inputs are *off*. In our design, these inputs are *off* when the clock is *off*. When the clock is *on*, the set signal is *on* if the calculation stage outputs *on*, setting the flip-flop *on*. The reset signal is *on* if the calculation stage outputs *off*, resetting the flip-flop *off*. Memory is required so the inputs to the other stages are not affected by the calculation of new local or neighbor states. **b** Circuit for our reaction–diffusion flip-flop using modules from Fig. 2. Copy modules ensure output is not depleted by internal feedback or downstream load

until a global clock signal turns *on*. For a given CA, the clock period must be designed so that all cells finish communicating and calculating before the clock signal turns *on*. The next state must be committed to memory before the clock turns *off*.

To ensure that calculated next states are not stored in memory unless the clock signal is *on*, an AND gate takes the clock signal and the calculated next state from the Calculation stage as inputs, and sends a set signal to the flip-flop module only when both the clock and the calculated next state are *on*. Another AND gate takes the clock signal and the inverse of the calculated next state as inputs, and produces a reset signal when the clock is *on* but the next state is *off*. The process of storing the new state in memory ends when the clock signal returns to a low value at the beginning of the next stage of computation.

4 Simulation of a reaction–diffusion cellular automaton

The complete automaton circuit is shown in Fig. 7. We simulated the simultaneous mass action reaction kinetics and diffusion for the entire system, using Rules 60 and 110 in the logic stage, and observed the intended cell updating for both rules (Fig. 9). The complete set of chemical reactions and the corresponding coupled partial differential equations describing these systems are provided in Section 8, along with all other simulation details. Concentrations within $[0, 0.3]\,\mu$M were considered *off*, while concentrations within $[0.7, 1]\,\mu$M were considered *on*. One irregularity that appears in our system is that the cells have blurred boundaries, an artifact that arises when chemical species produced inside of a cell diffuse across the cell boundary. This blurring effect is the reason that we included short spacer regions to separate adjacent cells, so that the logic inside of one cell does not interfere with the logic inside of its neighbors.

Two important parameters can break the reaction–diffusion program if not tuned carefully: the *on* time or ‘duty cycle’ of the clock signal, and the kinetic rates for the broadcast module. If the duty cycle is too short, then the flip-flop does not have enough time to store the intended next-generation state. In our simulations, this occurs for duty cycles shorter than 15–20 min. Particularly long duty cycles can also cause problems because the calculation stage of our CA is continually active while the clock is *on*. If the communication stage of a cell continues too long after the calculation of the cell’s next state is complete, the cell will continue to recalculate new values for the next state without waiting for the communication stage to receive updated values from neighboring cells, resulting in errors. In our simulations, duty cycles longer than about an hour and a half led cells to become desynchronized.

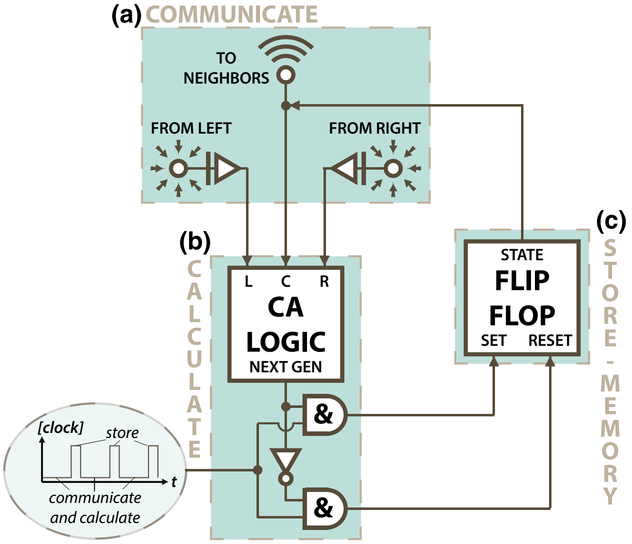

Fig. 7 CA circuit diagram **a** The ‘Communication’ stage. Current cell states are broadcast to neighbors, while neighbor states are received. **b** The ‘Calculation’ stage. The states of a cell and its neighbors are passed through a subcircuit that performs the update logic. The output from this subcircuit is *on* if the cell should change into an *on* state in the next generation. This next state is prevented from affecting the Memory stage by AND gates when the clock is *off*. **c** When the clock turns *on*, the next state is stored in the ‘Memory’ stage

The second critical parameter is the production rate constant for the broadcast module. When a cell is *on*, this constant must be high enough to saturate its neighbors with signaling molecule. In the worst case, where a cell is at the minimum *on* concentration of 0.7 μM, it must maintain a broadcast signal above the receive module’s threshold concentration at the farthest edge of its neighboring cell regions. On the other hand, when a cell is *off*, this constant must be low enough to avoid broadcasting any signal to its neighbors. Specifically, in the worst case where a cell is at the maximum *off* concentration of 0.3 μM, it must maintain a broadcast signal below the receive module’s threshold concentration at the closest edge of its neighboring cell regions. If either of these conditions are not met, then erroneous signals can be sent between cells.

5 Asynchronous cellular automata

One limitation to the design for the cellular automaton that we presented above is that it requires a global “clock” signal that is continuously supplied such that it continually changes concentration in an oscillatory pattern. It is likely that providing such a signal externally would be challenging, because existing biomolecular oscillators are generally noisy, and may not provide the required synchronization (Montagne et al. 2011). An alternative to the design we have presented would be to use an asynchronous cellular automaton, which does not require a clock signal to operate. Instead, for each update round, all cells in the network wait until they have received information indicating that their neighbors are ready to proceed to the next round, thus preventing any single cell from getting out of synchrony with its immediate neighbors.

A design for an asynchronous CA is outlined in Fig. 8. The circuit elements (i.e. modules) are the same as those for the synchronous CA, but more elements are required and they are arranged differently. In this asynchronous CA, three banks of memory modules are used to (1) store the cell's current state, (2) store the all of the received neighbor states, and (3) store the calculated next states. Between each memory bank, Boolean logic circuits inside of each cell prevent it from updating the next bank until the cell has received indication that its neighbor cells have updated the current bank. Cells can be at most one step ahead of their neighbors before they are forced to stop and wait for their neighbors to catch up. This strategy guarantees that each cell will pass through the correct series of states, and that local groups can never be out of synchronization. However, because each cell is permitted to proceed up to one step in advance of its neighbors, any two cells that are *N* neighbors away from each other can also be up to *N* time steps out of synchronization. So while the overall process will compute the correct dynamics for each individual cell, the state of the system may not represent the particular global state of the ideal cellular automaton for any particular time step. This issue could make it difficult to use such a CA to process incoming spatial information within the environment, because the computation process in different regions of the CA will not see this environmental information at the same stage of computation. Furthermore, due to the additional storage of information and verification circuitry, the overall circuit size of our asynchronous CA is much larger than the synchronous (clocked) design. However, we believe that an asynchronous automaton could overcome many of the disadvantages of the synchronous automaton we presented, because the asynchronous version requires less global coordination.

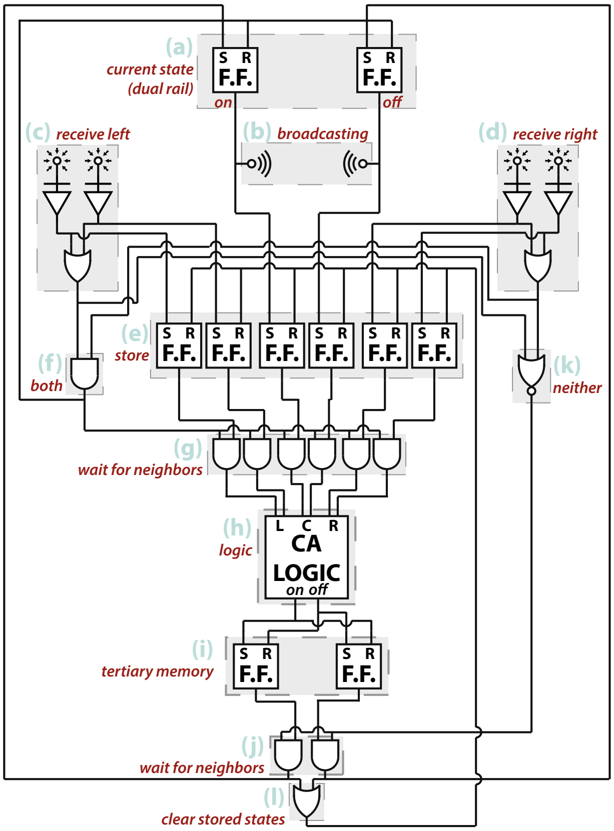

Fig. 8 An asynchronous cellular automaton. *a* A cell's state is stored in memory. Dual-rail logic is used to differentiate between neighbors that are *off* and neighbors that have had insufficient time to broadcast their state. **b** Current states are broadcast. *c* and *d* Neighbor states are received. *e* Dual-rail states of cell and both neighbors are stored in a secondary memory bank. This enables cells to remember neighbors states even after neighbors stop broadcasting. *f* When both neighbor broadcasts are received, a cell resets its primary memory, turns off its broadcast signals, and *g* passes the stored information to (*h*) the cellular automaton logic stage. If the time scale for chemical reactions t_R is much faster than the time scale for diffusive propagation between cells t_D, then the logic stage is guaranteed to have sufficient time to use the information stored in (*e*) to calculate the next state and store the results in (*i*) a tertiary memory bank before it stops receiving neighbor broadcasts. This calculated next state is blocked by (*j*) two AND gates until (*k*) all neighbor broadcasts have had time to turn off. Then, the secondary memory in (*e*) is reset by (*l*), the next state is stored in (*a*), and the cycle repeats

6 Discussion

In this work we develop a method for building a CA using a reaction–diffusion program. This program consists of a set of molecules that everywhere can react and diffuse in the same way, along with a small set of molecules that are patterned into a grid of cells. The collective actions of these molecules cause the pattern of molecules that encode an "on" state to change over time to display a series of patterns that are the successive states of a one-dimensional binary CA. While the construction we propose is for a one-dimensional binary CA, straightforward extensions to the system could be used to produce two- or three-dimensional CA, or CA in which cells can be in more than two states.

This construction thus suggests two important new capabilities for systems that are driven by designed chemical reaction networks. First, this system provides a way to generate dynamic spatial patterns, where the concentrations of species vary continuously over time, by encoding these dynamics within a CA. Second, this system makes it possible

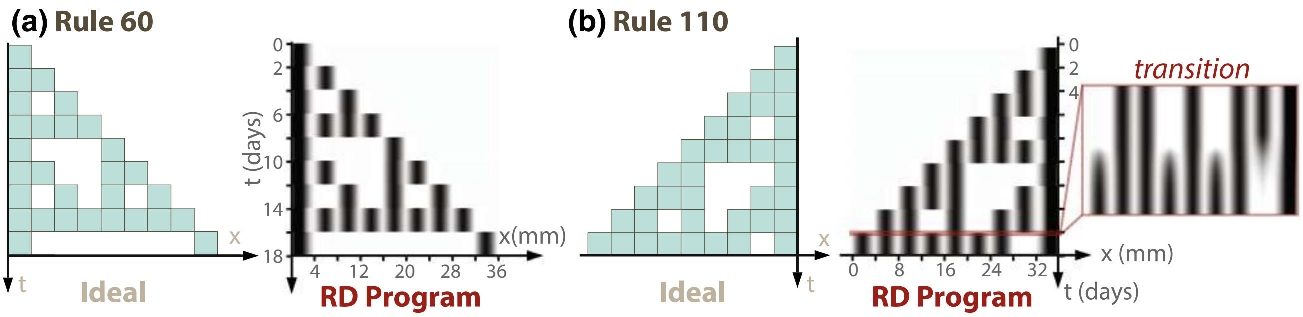

Fig. 9 *Results of chemical CA simulations* Ideal CA (*left*) compared to our simulated reaction–diffusion program (*right*). Every three-length binary input state is contained in each pattern, demonstrating correct updating for all eight possible local states. **a** Rule 60. **b** Rule 110. The dynamics shown here were computed using the set of coupled partial differential equations in Section 8. The detail of the rapid dynamics of a state transition are shown on the far right

to perform computations using molecules in which the same molecular species simultaneously perform different computations within different regions of a substrate.

The capacity for this kind of spatial computation is likely to be an important part of scaling the capacity for molecular systems to process information. Because the number of independent molecular interfaces is inherently limited, it is not possible to arbitrarily increase the number of interacting molecules within a well-mixed system without introducing crosstalk. The importance of spatial computation with molecules is underscored by its prevalence in living systems. Reaction–diffusion processes are used for signaling within cells, and across tissues, where different cells (which each share the same genome) collectively coordinate tissue behavior.

While other molecular processes can perform Turing universal computation with only a limited number of molecular species, i.e. they are uniform, these constructions are highly susceptible to errors resulting from random fluctuations inherent in stochastic chemical kinetics. For example, they may require that the state of a computation be encoded within a single molecular assembly, as in algorithmic self-assembly (Rothemund et al. 2004) or store information in the form of the precise number of molecules in the system (Soloveichik et al. 2008; Qian et al. 2011). In these systems, a computation may be irrevocably corrupted by a single anomalous chemical interaction. In contrast, computation by the CA that we describe involves the collective action of many molecules, so it is not susceptible to errors caused by a small number of microscopic events.

However, the designs presented in this paper require the construction of large chemical reaction networks, a clock signal at regular intervals and a printed grid of different “key” molecules. Our reaction network uses 65 species to emulate a “Rule 60” CA, and 76 species to emulate a “Rule 110” CA. Further emulating these abstract chemical networks using DNA strand-displacement reactions could increase the network size by an order of magnitude, because multiple intermediate DNA strands are generally required when emulating reactions. Likely there are simplifications that could be made to our circuit, as our goal was to demonstrate that such an implementation is theoretically possible instead of designing the smallest possible circuit. For instance, it may be possible to condense some sections of our system into smaller special case circuits for particular CA updating rules. Additionally, our four-key system that provides unique identities to cells in a local group is expensive in terms of number of species, requiring four separate sets of transmitter modules and eight separate sets of receiver modules in one-dimensional space, and a more clever means for identifying neighboring cells may exist. However, it is unclear how to reduce the number of strands in our system by an order of magnitude.

Generally, the complexity of our circuits suggests that implementing even a simple one-dimensional automaton would be challenging with current chemical computers. Constructing a CA as complex as von Neumann’s self-replicating automata is likely to be infeasible for the foreseeable future. It will therefore be important to ask whether there are more efficient models for spatial computing in which complex behaviors such as self-replication or healing can be designed as simply as possible. One starting point is to consider computation systems that do not require an explicit regular grid, such as Lindenmayer systems (Lindenmayer 1968), or graph automata (Wu and Rosenfeld 1979; Tomita et al. 2002), and un-clocked systems such as asynchronous CA (Nehaniv 2004).

More generally, we might ask not only how to perform molecular computation using space as a medium, but how to construct a scalable architecture for computing appropriate responses of a material to stimuli that are presented across space and time. Patterns generated by CA could act as blueprints, encoding dynamic information spatially. By constructing CA in chemical networks, it may be possible to use this information to coordinate the behavior of intelligent programmable materials. Biological cells, tissues, and synthetic nanostructures could potentially respond to local instructions released by an embedded chemical automaton. A CA could endow these physical systems with unique properties, creating artificial structures that heal, self-replicate and evolve.

Acknowledgments The authors would like to thank Paul Rothemund, Damien Woods, Josh Fern, John Zenk, and the anonymous referees for insightful reading and comments. This work was supported by NSF-CCF-1161941 and a grant to the Turing Centenary Project by the John Templeton Foundation.

Rule 110 chemical reactions (synonymous with Sect. 8)

This section describes the complete set of abstract chemical reactions that govern our Rule 110 chemical automaton. These chemical reactions can be converted into a set of coupled partial differential equations and solved to observe how the system behaves over time (see Section 8). Figure 10 shows a detail of the circuit from Fig. 7 for a Rule 110 automaton, with each chemical species labeled.

1. Communication stage

   a. Broadcasting:

$$SrcA + keyA + Last \xrightarrow{k_B} SrcA + SigA + keyA + Last \tag{7.1}$$

$$SigA \xrightarrow{k_{Bd}} waste \tag{7.2}$$

$$SrcB + keyB + Last \xrightarrow{k_B} SrcB + SigB + keyB + Last \tag{7.3}$$

$$SigB \xrightarrow{k_{Bd}} waste \tag{7.4}$$

$$SrcC + keyC + Last \xrightarrow{k_B} SrcC + SigC + keyC + Last \tag{7.5}$$

$$SigC \xrightarrow{k_{Bd}} waste \tag{7.6}$$

$$SrcD + keyD + Last \xrightarrow{k_B} SrcD + SigD + keyD + Last \tag{7.7}$$

$$SigD \xrightarrow{k_{Bd}} waste \tag{7.8}$$

   a. Receiving and processing left-hand signal:

$$SigD + keyA \xrightarrow{k_x} SigD + Lraw + keyA \tag{7.9}$$

$$SigA + keyB \xrightarrow{k_x} SigA + Lraw + keyB \tag{7.10}$$

$$SigB + keyC \xrightarrow{k_x} SigB + Lraw + keyC \tag{7.11}$$

$$SigC + keyD \xrightarrow{k_x} SigC + Lraw + keyD \tag{7.12}$$

$$Lraw \xrightarrow{4*k_x} waste \tag{7.13}$$

$$source \xrightarrow{c_{recTh}*k_p} ThL \xrightarrow{k_x} waste \tag{7.14}$$

$$source \xrightarrow{k_p} AmpL \xrightarrow{k_x} waste \tag{7.15}$$

$$Lraw + ThL \xrightarrow{k_T} waste \tag{7.16}$$

$$Lraw + AmpL \xrightarrow{k_L} Lraw + Lft \tag{7.17}$$

$$Lft \xrightarrow{k_x} waste \tag{7.18}$$

   a. Receiving and processing right-hand signal:

$$SigB + keyA \xrightarrow{k_x} SigB + Rraw + keyA \tag{7.19}$$

$$SigC + keyB \xrightarrow{k_x} SigC + Rraw + keyB \tag{7.20}$$

$$SigD + keyC \xrightarrow{k_x} SigD + Rraw + keyC \tag{7.21}$$

$$SigA + keyD \xrightarrow{k_x} SigA + Rraw + keyD \tag{7.22}$$

$$Rraw \xrightarrow{4*k_x} waste \tag{7.23}$$

$$source \xrightarrow{c_{recTh}*k_p} Thr \xrightarrow{k_x} waste \tag{7.24}$$

$$source \xrightarrow{k_p} Ampr \xrightarrow{k_x} waste \tag{7.25}$$

$$Rraw + Thr \xrightarrow{k_T} waste \tag{7.26}$$

$$Rraw + Ampr \xrightarrow{k_L} Rraw + Rght \tag{7.27}$$

$$Rght \xrightarrow{k_x} waste \tag{7.28}$$

2. Calculation Stage

   a. Copy left, right and previous time step (pr) signals (multiple gates operate on each):

$$Last \xrightarrow{k_x} Last + BrPr \tag{7.29}$$

$$BrPr \xrightarrow{k_x} waste \tag{7.30}$$

$$Rght \xrightarrow{k_x} Rght + RBr \tag{7.31}$$

$$RBr \xrightarrow{k_x} waste \tag{7.32}$$

$$Last \xrightarrow{k_x} Last + OnPr \tag{7.33}$$

$$\mathit{OnPr} \xrightarrow{k_x} \mathit{waste} \tag{7.34}$$

$$\mathit{Rght} \xrightarrow{k_x} \mathit{Rght} + \mathit{RCr} \tag{7.35}$$

$$\mathit{RCr} \xrightarrow{k_x} \mathit{waste} \tag{7.36}$$

$$\mathit{Lft} \xrightarrow{k_x} \mathit{Lft} + \mathit{LCr} \tag{7.37}$$

$$\mathit{LCr} \xrightarrow{k_x} \mathit{waste} \tag{7.38}$$

$$\mathit{source} \xrightarrow{k_p} \mathit{OffPr} \tag{7.39}$$

$$\mathit{OffPr} \xrightarrow{k_x} \mathit{waste} \tag{7.40}$$

$$\mathit{BrPr} + \mathit{OffPr} \xrightarrow{k_T} \mathit{waste} \tag{7.41}$$

a. Boolean logic for Br (birth) condition: *off* to *on* transition:

$$\mathit{source} \xrightarrow{2{*}k_p} \mathit{SgBr} \xrightarrow{k_x} \mathit{waste} \tag{7.42}$$

$$\mathit{OffPr} + \mathit{SgBr} \xrightarrow{k_L} \mathit{SumBr} \xrightarrow{k_x} \mathit{waste} \tag{7.43}$$

$$\mathit{RBr} + \mathit{SgBr} \xrightarrow{k_L} \mathit{SumBr} \tag{7.44}$$

$$\mathit{source} \xrightarrow{1.35{*}k_p} \mathit{ThBr} \xrightarrow{k_x} \mathit{waste} \tag{7.45}$$

$$\mathit{source} \xrightarrow{k_p} \mathit{AmpBr} \xrightarrow{k_x} \mathit{waste} \tag{7.46}$$

$$\mathit{SumBr} + \mathit{ThBr} \xrightarrow{k_T} \mathit{waste} \tag{7.47}$$

$$\mathit{SumBr} + \mathit{AmpBr} \xrightarrow{k_L} \mathit{SumBr} + \mathit{Br} \tag{7.48}$$

$$\mathit{Br} \xrightarrow{k_x} \mathit{waste} \tag{7.49}$$

a. Boolean logic for no death condition (*on* and at least one neighbor *off*): stay on:

$$\mathit{source} \xrightarrow{2{*}k_p} \mathit{SgnbOff} \xrightarrow{k_x} \mathit{waste} \tag{7.50}$$

$$\mathit{RCr} + \mathit{SgnbOff} \xrightarrow{k_L} \mathit{SumnbOff} \xrightarrow{k_x} \mathit{waste} \tag{7.51}$$

$$\mathit{LCr} + \mathit{SgnbOff} \xrightarrow{k_L} \mathit{SumnbOff} \tag{7.52}$$

$$\mathit{source} \xrightarrow{1.35{*}k_p} \mathit{ThnbOff} \xrightarrow{k_x} \mathit{waste} \tag{7.53}$$

$$\mathit{source} \xrightarrow{k_p} \mathit{AmpnbOff} \xrightarrow{k_x} \mathit{waste} \tag{7.54}$$

$$\mathit{SumnbOff} + \mathit{ThnbOff} \xrightarrow{k_T} \mathit{waste} \tag{7.55}$$

$$\mathit{ThnbOff} + \mathit{AmpnbOff} \xrightarrow{k_L} \mathit{ThnbOff} + \mathit{nbrOff} \tag{7.56}$$

$$\mathit{nbrOff} \xrightarrow{k_x} \mathit{waste} \tag{7.57}$$

$$\mathit{source} \xrightarrow{2{*}k_p} \mathit{SgCr} \xrightarrow{k_x} \mathit{waste} \tag{7.58}$$

$$\mathit{OnPr} + \mathit{SgCr} \xrightarrow{k_L} \mathit{SumCr} \xrightarrow{k_x} \mathit{waste} \tag{7.59}$$

$$\mathit{nbrOff} + \mathit{SgCr} \xrightarrow{k_L} \mathit{SumCr} \tag{7.60}$$

$$\mathit{source} \xrightarrow{1.35{*}k_p} \mathit{ThCr} \xrightarrow{k_x} \mathit{waste} \tag{7.61}$$

$$\mathit{source} \xrightarrow{k_p} \mathit{AmpCr} \xrightarrow{k_x} \mathit{waste} \tag{7.62}$$

$$\mathit{SumCr} + \mathit{ThCr} \xrightarrow{k_T} \mathit{waste} \tag{7.63}$$

$$\mathit{SumCr} + \mathit{AmpCr} \xrightarrow{k_L} \mathit{SumCr} + \mathit{Live} \tag{7.64}$$

$$\mathit{Live} \xrightarrow{k_x} \mathit{waste} \tag{7.65}$$

a. Boolean logic to determine if next state is *on*:

$$\mathit{source} \xrightarrow{2{*}k_p} \mathit{SgNx} \xrightarrow{k_x} \mathit{waste} \tag{7.66}$$

$$\mathit{Live} + \mathit{SgNx} \overset{k_L}{\to} \mathit{SumNx} \overset{k_x}{\to} \mathit{waste} \tag{7.67}$$

$$\mathit{Br} + \mathit{SgNx} \overset{k_L}{\to} \mathit{SumNx} \tag{7.68}$$

$$\mathit{source} \xrightarrow{0.65{*}k_p} \mathit{ThNx} \overset{k_x}{\to} \mathit{waste} \tag{7.69}$$

$$\mathit{source} \overset{k_p}{\to} \mathit{AmpNx} \overset{k_x}{\to} \mathit{waste} \tag{7.70}$$

$$\mathit{SumNx} + \mathit{ThNx} \overset{k_T}{\to} \mathit{waste} \tag{7.71}$$

$$\mathit{SumNx} + \mathit{AmpNx} \overset{k_L}{\to} \mathit{SumNx} + \mathit{Nx} \tag{7.72}$$

$$\mathit{Nx} \overset{k_x}{\to} \mathit{waste} \tag{7.73}$$

$$\mathit{Nx} \overset{k_x}{\to} \mathit{Nx} + \mathit{OnNx} \tag{7.74}$$

$$\mathit{OnNx} \overset{k_x}{\to} \mathit{waste} \tag{7.75}$$

a. Boolean logic to determine if next state is *off*:

$$\mathit{Nx} \overset{k_x}{\to} \mathit{Nx} + \mathit{NtNx} \tag{7.76}$$

$$\mathit{NtNx} \overset{k_x}{\to} \mathit{waste} \tag{7.77}$$

$$\mathit{source} \overset{k_p}{\to} \mathit{OffNx} \overset{k_x}{\to} \mathit{waste} \tag{7.78}$$

$$\mathit{NtNx} + \mathit{OffNx} \overset{k_T}{\to} \mathit{waste} \tag{7.79}$$

a. Clocked synchronization gates:

$$\mathit{clk} \overset{k_x}{\to} \mathit{clk} + \mathit{clkOn} \tag{7.80}$$

$$clkOn \xrightarrow{k_x} waste \tag{7.81}$$

$$clk \xrightarrow{k_x} clk + clkOff \tag{7.82}$$

$$clkOff \xrightarrow{k_x} waste \tag{7.83}$$

$$source \xrightarrow{2*k_p} SgNxOn \xrightarrow{k_x} waste \tag{7.84}$$

$$OnNx + SgNxOn \xrightarrow{k_L} OnNxSum \xrightarrow{k_x} waste \tag{7.85}$$

$$clkOn + SgNxOn \xrightarrow{k_L} OnNxSum \tag{7.86}$$

$$source \xrightarrow{1.35*k_p} OnNxTh \xrightarrow{k_x} waste \tag{7.87}$$

$$source \xrightarrow{k_p} OnNxAmp \xrightarrow{k_x} waste \tag{7.88}$$

$$OnNxSum + OnNxTh \xrightarrow{k_T} waste \tag{7.89}$$

$$OnNxSum + OnNxAmp \xrightarrow{k_L} OnNxSum + SetBfr \tag{7.90}$$

$$SetBfr \xrightarrow{k_x} waste \tag{7.91}$$

$$source \xrightarrow{2*k_p} SgOffNx \xrightarrow{k_x} waste \tag{7.92}$$

$$OffNx + SgOffNx \xrightarrow{k_L} SumOffNx \xrightarrow{k_x} waste \tag{7.93}$$

$$clkOff + SgOffNx \xrightarrow{k_L} SumOffNx \tag{7.94}$$

$$source \xrightarrow{1.35*k_p} ThOffNx \xrightarrow{k_x} waste \tag{7.95}$$

$$source \xrightarrow{k_p} AmpOffNx \xrightarrow{k_x} waste \tag{7.96}$$

$$SumOffNx + ThOffNx \xrightarrow{k_T} waste \tag{7.97}$$

$$SumOffNx + AmpOffNx \xrightarrow{k_L} SumOffNx + ResBfr \tag{7.98}$$

$$ResBfr \xrightarrow{k_x} waste \tag{7.99}$$

3. Storage Stage

   a. Copies of Set/Res signals:

$$SetBfr \xrightarrow{k_x} SetBfr + Set \tag{7.100}$$

$$Set \xrightarrow{k_x} waste \tag{7.101}$$

$$ResBfr \xrightarrow{k_x} ResBfr + Res \tag{7.102}$$

$$Res \xrightarrow{k_x} waste \tag{7.103}$$

   a. Flip-flop module:

$$ffBfrd \xrightarrow{k_x} ffBfrd + ffFback \tag{7.104}$$

$$ffFback \xrightarrow{k_x} waste \tag{7.105}$$

$$source \xrightarrow{2*k_p} N1Sg \xrightarrow{k_x} waste \tag{7.106}$$

$$Set + N1Sg \xrightarrow{k_L} N1Sum \xrightarrow{k_x} waste \tag{7.107}$$

$$ffFback + N1Sg \xrightarrow{k_L} N1Sum \tag{7.108}$$

$$source \xrightarrow{0.65*k_p} N1Th \xrightarrow{k_x} waste \tag{7.109}$$

$$source \xrightarrow{k_p} N1 \xrightarrow{k_x} waste \tag{7.110}$$

$$N1Sum + N1Th \xrightarrow{k_T} waste \tag{7.111}$$

$$N1Th + N1 \xrightarrow{k_L} N1Th + ffFbackNot \tag{7.112}$$

$$ffFbackNot \xrightarrow{k_x} waste \tag{7.113}$$

$$source \xrightarrow{2*k_p} N2Sg \xrightarrow{k_x} waste \tag{7.114}$$

$$Res + N2Sg \xrightarrow{k_L} N2Sum \xrightarrow{k_x} waste \tag{7.115}$$

$$ffFbackNot + N2Sg \xrightarrow{k_L} N2Sum \tag{7.116}$$

$$source \xrightarrow{0.65*k_p} N2Th \xrightarrow{k_x} waste \tag{7.117}$$

$$source \xrightarrow{k_p} N2 \xrightarrow{k_x} waste \tag{7.118}$$

$$N2Sum + N2Th \xrightarrow{k_T} waste \tag{7.119}$$

$$N2Th + N2 \xrightarrow{k_L} N2Th + ffBfrd \tag{7.120}$$

$$ffBfrd \xrightarrow{k_x} waste \tag{7.121}$$

$$ffBfrd \xrightarrow{k_x} ffBfrd + Last \tag{7.122}$$

$$Last \xrightarrow{k_x} waste \tag{7.123}$$

Rule 110 partial differential equations

> "I hope to say something about a 'continuous' rather than 'crystalline' model [of automata]. There, as far as I can now see, a system of nonlinear partial differential equations, essentially of the diffusion type, will be used." John von Neumann (Oct. 28th, 1952) discussing unfinished Theory of Automata.
>
> von Neumann papers, Library of Congress, Box 28 "Theory of Automata".

This section describes the set of coupled partial differential equations that govern our Rule 110 chemical automaton, derived from the reactions in this section. These equations use standard mass-action equations and the diffusion equation.

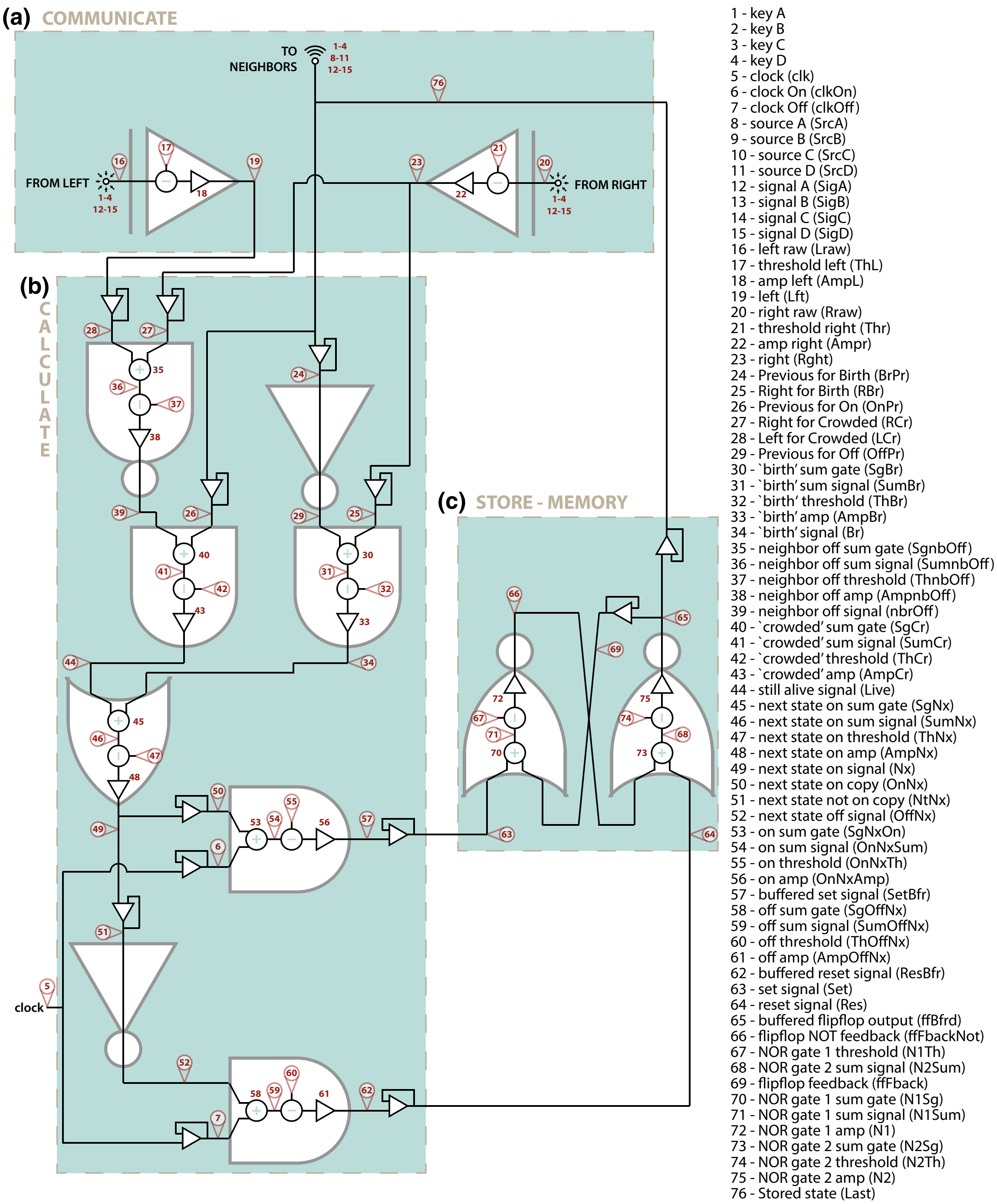
(a) COMMUNICATE
TO NEIGHBORS
1-4
8-11
12-15
FROM LEFT
FROM RIGHT
(b) CALCULATE
(c) STORE - MEMORY
clock
1 - key A
2 - key B
3 - key C
4 - key D
5 - clock (clk)
6 - clock On (clkOn)
7 - clock Off (clkOff)
8 - source A (SrcA)
9 - source B (SrcB)
10 - source C (SrcC)
11 - source D (SrcD)
12 - signal A (SigA)
13 - signal B (SigB)
14 - signal C (SigC)
15 - signal D (SigD)
16 - left raw (Lraw)
17 - threshold left (ThL)
18 - amp left (AmpL)
19 - left (Lft)
20 - right raw (Rraw)
21 - threshold right (Thr)
22 - amp right (Ampr)
23 - right (Rght)
24 - Previous for Birth (BrPr)
25 - Right for Birth (RBr)
26 - Previous for On (OnPr)
27 - Right for Crowded (RCr)
28 - Left for Crowded (LCr)
29 - Previous for Off (OffPr)
30 - `birth' sum gate (SgBr)
31 - `birth' sum signal (SumBr)
32 - `birth' threshold (ThBr)
33 - `birth' amp (AmpBr)
34 - `birth' signal (Br)
35 - neighbor off sum gate (SgnbOff)
36 - neighbor off sum signal (SumnbOff)
37 - neighbor off threshold (ThnbOff)
38 - neighbor off amp (AmpnbOff)
39 - neighbor off signal (nbrOff)
40 - `crowded' sum gate (SgCr)
41 - `crowded' sum signal (SumCr)
42 - `crowded' threshold (ThCr)
43 - `crowded' amp (AmpCr)
44 - still alive signal (Live)
45 - next state on sum gate (SgNx)
46 - next state on sum signal (SumNx)
47 - next state on threshold (ThNx)
48 - next state on amp (AmpNx)
49 - next state on signal (Nx)
50 - next state on copy (OnNx)
51 - next state not on copy (NtNx)
52 - next state off signal (OffNx)
53 - on sum gate (SgNxOn)
54 - on sum signal (OnNxSum)
55 - on threshold (OnNxTh)
56 - on amp (OnNxAmp)
57 - buffered set signal (SetBfr)
58 - off sum gate (SgOffNx)
59 - off sum signal (SumOffNx)
60 - off threshold (ThOffNx)
61 - off amp (AmpOffNx)
62 - buffered reset signal (ResBfr)
63 - set signal (Set)
64 - reset signal (Res)
65 - buffered flipflop output (ffBfrd)
66 - flipflop NOT feedback (ffFbackNot)
67 - NOR gate 1 threshold (N1Th)
68 - NOR gate 2 sum signal (N2Sum)
69 - flipflop feedback (ffFback)
70 - NOR gate 1 sum gate (N1Sg)
71 - NOR gate 1 sum signal (N1Sum)
72 - NOR gate 1 amp (N1)
73 - NOR gate 2 sum gate (N2Sg)
74 - NOR gate 2 threshold (N2Th)
75 - NOR gate 2 amp (N2)
76 - Stored state (Last)

Fig. 10 Chemical reaction–diffusion circuit for Rule 110. This is a detailed version of the circuit outlined in Fig. 7, using the modules defined in Fig. 2. Species are labelled in red by their equation numbers from Section 8., with species names and abbreviations to the right

Figure 9b contains a plot of the solution to these equations. One equation is devoted to each of the 76 species in our network. Figure 10 shows a detail of the circuit from Fig. 7 for a Rule 110 automaton, with each species labeled.

Unless otherwise specified, all species start with zero initial concentration. Absorbing boundary conditions apply to all species whose concentrations change over time. Our reaction rate constants and diffusion coefficients are selected to be realistically attainable values for DNA-based reaction–diffusion networks, on the same order of magnitude as experimentally derived data in the literature (Zhang and Winfree 2009; Lukacs et al. 2000; Stellwagen et al. 2003). The *Mathematica* code we used to numerically solve these equations is available upon request. Constants:

$$\begin{array}{ccc}
xMax = 64 & k_p = 0.002\ \mu\mathrm{M\,s}^{-1} & k_{Bd} = 0.00002\,\mathrm{s}^{-1} \\
D = 0.00015\ \mathrm{mm}^2\,\mathrm{s}^{-1} & k_d = 0.002\ \mu\,\mathrm{s}^{-1} & c_{recTh} = 0.5\ \mu\mathrm{M} \\
k_T = 20\ \mu\mathrm{M}^{-1}\,\mathrm{s}^{-1} & k_x = 0.002\ \mu\mathrm{M}^{-1}\,\mathrm{s}^{-1} & clkPeriod = 2*24*3600\ \mathrm{s} \\
k_L = 0.2\ \mu\mathrm{M}^{-1}\,\mathrm{s}^{-1} & k_B = 0.0002\ \mu\mathrm{M}^{-2}\,\mathrm{s}^{-1} & clkDuty = .5*3600\ \mathrm{s}
\end{array}$$

1. External signals:

   a. Keys (in μM):

$$Key_A(t,x) = \begin{cases} 1/(1+Exp[-25*(Mod(x,16)-1)]) & : Mod(x,16) \le 2 \\ 1-1/(1+Exp[-25*(Mod(x,16)-3)]) & : otherwise. \end{cases} \tag{8.1}$$

$$Key_B(t,x) = \begin{cases} 1/(1+Exp[-25*(Mod(x,16)-5)]) & : Mod(x,16) \le 6 \\ 1-1/(1+Exp[-25*(Mod(x,16)-7)]) & : otherwise. \end{cases} \tag{8.2}$$

$$Key_C(t,x) = \begin{cases} 1/(1+Exp[-25(Mod(x,16)-9)]) & : mod(x,16) \le 10 \\ 1-1/(1+Exp[-25(Mod(x,16)-11)]) & : otherwise. \end{cases} \tag{8.3}$$

$$Key_D(t,x) = \begin{cases} 1/(1+Exp[-25(Mod(x,16)-13)]) & : mod(x,16) \le 14 \\ 1-1/(1+Exp[-25(Mod(x,16)-15)]) & : otherwise. \end{cases} \tag{8.4}$$

   b. Clock:

$$clk(t,x) = \begin{cases} 1\ \mu\mathrm{M} & : t < 4000 \\ 1\ \mu\mathrm{M} & : Mod(t, clkPeriod) < clkDuty \\ 0 & : otherwise. \end{cases} \tag{8.5}$$

$$\begin{aligned}\frac{\partial clk_{On}(t,x)}{\partial t} = {} & D\nabla^2 clk_{On}(t,x) - k_d \\ & * clk_{On}(t,x) + k_d * clk(t,x) \\ & - k_L * clk_{On}(t,x) * Sg_{nxOn}(t,x)\end{aligned} \tag{8.6}$$

$$\begin{aligned}\frac{\partial clk_{Off}(t,x)}{\partial t} = {} & D\nabla^2 clk_{Off}(t,x) - k_d * clk_{Off}(t,x) \\ & + k_d * clk(t,x) - k_L * clk_{Off}(t,x) \\ & * Sg_{OffNx}(t,x)\end{aligned} \tag{8.7}$$

2. Communication Stage:

   a. Broadcast Modules (in μM):

$$Src_A(t,x) = 1 \tag{8.8}$$

$$Src_B(t,x) = 1 \tag{8.9}$$

$$Src_C(t,x) = 1 \tag{8.10}$$

$$Src_D(t,x) = 1 \tag{8.11}$$

   b. Broadcast signals, note initial concentration of Sig_A triggers initial *on* cell:

$$\begin{aligned}\frac{\partial Sig_A(t,x)}{\partial t} = {} & D\nabla^2 Sig_A(t,x) + k_B Src_A(t,x) Key_A \\ & (t,x) Last(t,x) - k_{Bd} Sig_A(t,x) \\ Sig_A(t=0,x) = {} & \begin{cases} 2\ \mu\mathrm{M} & : x_{max} - 8 < x < x_{max} - 0.1 \\ 0 & : otherwise. \end{cases}\end{aligned} \tag{8.12}$$

$$\begin{aligned}\frac{\partial Sig_B(t,x)}{\partial t} = {} & D\nabla^2 Sig_B(t,x) + k_B Src_B(t,x) Key_B(t,x) \\ & \times Last(t,x) - k_{Bd} Sig_B(t,x)\end{aligned} \tag{8.13}$$

$$\begin{aligned}\frac{\partial Sig_C(t,x)}{\partial t} = {} & D\nabla^2 Sig_C(t,x) + k_B Src_C(t,x) \\ & \times Key_C(t,x) Last(t,x) - k_{Bd} Sig_C(t,x)\end{aligned} \tag{8.14}$$

$$\begin{aligned}\frac{\partial Sig_D(t,x)}{\partial t} = {} & D\nabla^2 Sig_D(t,x) + k_B Src_D(t,x) \\ & \times Key_D(t,x) Last(t,x) - k_{Bd} Sig_D(t,x)\end{aligned} \tag{8.15}$$

   c. Receiving and processing left-hand neighbor signal:

$$\begin{aligned}\frac{\partial L_{raw}(t,x)}{\partial t} = {} & D\nabla^2 L_{raw}(t,x) - 4k_d L_{raw}(t,x) \\ & - k_T L_{raw}(t,x) Th_l(t,x) + k_x Sig_D(t,x) Key_A(t,x) \\ & + k_x Sig_A(t,x) Key_B(t,x) + k_x Sig_B(t,x) Key_C(t,x) \\ & + k_x Sig_C(t,x) Key_D(t,x)\end{aligned} \tag{8.16}$$

$$\begin{aligned}\frac{\partial Th_l(t,x)}{\partial t} &= D\nabla^2 Th_l(t,x) + c_{recTh}k_p - k_d Th_l(t,x)\\ &\quad - k_T L_{raw}(t,x)Th_l(t,x)\end{aligned}\tag{8.17}$$

$$\begin{aligned}\frac{\partial Amp_l(t,x)}{\partial t} &= D\nabla^2 Amp_l(t,x) + k_p - k_d Amp_l(t,x)\\ &\quad - k_L L_{raw}(t,x)Amp_l(t,x)\end{aligned}\tag{8.18}$$

$$\begin{aligned}\frac{\partial Lft(t,x)}{\partial t} &= D\nabla^2 Lft(t,x) - k_d Lft(t,x)\\ &\quad + k_L L_{raw}(t,x)Amp_l(t,x)\end{aligned}\tag{8.19}$$

d. Receiving and processing right-hand neighbor signal:

$$\begin{aligned}\frac{\partial R_{raw}(t,x)}{\partial t} &= D\nabla^2 R_{raw}(t,x) - 4k_d R_{raw}(t,x)\\ &\quad - k_T R_{raw}(t,x)Th_r(t,x) + k_x Sig_{(}t,x)Key_A(t,x)\\ &\quad + k_x Sig_C(t,x)Key_B(t,x) + k_x Sig_D(t,x)Key_C(t,x)\\ &\quad + k_x Sig_A(t,x)Key_D(t,x)\end{aligned}\tag{8.20}$$

$$\begin{aligned}\frac{\partial Th_r(t,x)}{\partial t} &= D\nabla^2 Th_r(t,x) + c_{recTh}k_p\\ &\quad - k_d Th_r(t,x) - k_T R_{raw}(t,x)Th_r(t,x)\end{aligned}\tag{8.21}$$

$$\begin{aligned}\frac{\partial Amp_{rt}(t,x)}{\partial t} &= D\nabla^2 Amp_{rt}(t,x) + k_p\\ &\quad - k_d Amp_{rt}(t,x) - k_L R_{raw}(t,x)Amp_{rt}(t,x)\end{aligned}\tag{8.22}$$

$$\begin{aligned}\frac{\partial Rght(t,x)}{\partial t} &= D\nabla^2 Rght(t,x) - k_d Rght(t,x)\\ &\quad + k_L R_{raw}(t,x)Amp_{rt}(t,x)\end{aligned}\tag{8.23}$$

3. Calculation Stage:

a. Copy left, right and previous time step (pr) signals (multiple gates operate on each)

$$\begin{aligned}\frac{\partial Br_{pr}(t,x)}{\partial t} &= D\nabla^2 Br_{pr}(t,x) - k_d Br_{pr}(t,x)\\ &\quad + k_d Last(t,x) - k_T Br_{pr}(t,x)Off_{pr}(t,x)\end{aligned}\tag{8.24}$$

$$\begin{aligned}\frac{\partial R_{br}(t,x)}{\partial t} &= D\nabla^2 R_{br}(t,x) - k_d R_{br}(t,x)\\ &\quad + k_d Rght(t,x) - k_L R_{br}(t,x)Sg_{br}(t,x)\end{aligned}\tag{8.25}$$

$$\begin{aligned}\frac{\partial On_{pr}(t,x)}{\partial t} &= D\nabla^2 On_{pr}(t,x) - k_d On_{pr}(t,x)\\ &\quad + k_d Last(t,x) - k_L On_{pr}(t,x)Sg_{cr}(t,x)\end{aligned}\tag{8.26}$$

$$\begin{aligned}\frac{\partial R_{cr}(t,x)}{\partial t} &= D\nabla^2 R_{cr}(t,x) - k_d R_{cr}(t,x)\\ &\quad + k_d Rght(t,x) - k_L R_{cr}(t,x)Sg_{nbOff}(t,x)\end{aligned}\tag{8.27}$$

$$\begin{aligned}\frac{\partial L_{cr}(t,x)}{\partial t} &= D\nabla^2 L_{cr}(t,x) - k_d L_{cr}(t,x)\\ &\quad + k_d Lft(t,x) - k_L L_{cr}(t,x)Sg_{nbOff}(t,x)\end{aligned}\tag{8.28}$$

$$\begin{aligned}\frac{\partial Off_{pr}(t,x)}{\partial t} &= D\nabla^2 Off_{pr}(t,x) + k_p - k_d Off_{pr}(t,x)\\ &\quad -_T Br_{pr}(t,x)Off_{pr}(t,x) - k_L Off_{pr}(t,x)Sg_{br}(t,x)\end{aligned}\tag{8.29}$$

b. Boolean logic for Br (birth) condition: *off* to *on* transition

$$\begin{aligned}\frac{\partial Sg_{br}(t,x)}{\partial t} &= D\nabla^2 Sg_{br}(t,x) + 2k_p - k_d Sg_{br}(t,x)\\ &\quad - k_L Off_{pr}(t,x)Sg_{br}(t,x) - k_L R_{br}(t,x)Sg_{br}(t,x)\end{aligned}\tag{8.30}$$

$$\begin{aligned}\frac{\partial Sum_{br}(t,x)}{\partial t} &= D\nabla^2 Sum_{br}(t,x) - k_d Sum_{br}(t,x)\\ &\quad + k_L Off_{pr}(t,x)Sg_{br}(t,x) + k_L R_{br}(t,x)Sg_{br}(t,x)\\ &\quad - k_T Sum_{br}(t,x)Th_{br}(t,x)\end{aligned}\tag{8.31}$$

$$\begin{aligned}\frac{\partial Th_{br}(t,x)}{\partial t} &= D\nabla^2 Th_{br}(t,x) + 1.35k_p - k_d Th_{br}(t,x)\\ &\quad - k_T Sum_{br}(t,x)Th_{br}(t,x)\end{aligned}\tag{8.32}$$

$$\begin{aligned}\frac{\partial Amp_{br}(t,x)}{\partial t} &= D\nabla^2 Amp_{br}(t,x) + k_p\\ &\quad - k_d Amp_{br}(t,x) - k_L Sum_{br}(t,x)\\ &\quad \times Amp_{br}(t,x)\end{aligned}\tag{8.33}$$

$$\begin{aligned}\frac{\partial Br(t,x)}{\partial t} &= D\nabla^2 Br(t,x) - k_d Br(t,x)\\ &\quad + k_L Sum_{br}(t,x)Amp_{br}(t,x) - k_L Br(t,x)Sg_{nx}(t,x)\end{aligned}\tag{8.34}$$

c. Boolean logic for no death condition (*on* and at least one neighbor *off*): stay *on*

$$\begin{aligned}\frac{\partial Sg_{nbOff}(t,x)}{\partial t} &= D\nabla^2 Sg_{nbOff}(t,x) + 2k_p\\ &\quad - k_d Sg_{nbOff}(t,x) - k_L R_{cr}(t,x)Sg_{nbOff}(t,x)\\ &\quad - k_L L_{cr}(t,x)Sg_{nbOff}(t,x)\end{aligned}\tag{8.35}$$

$$\begin{aligned}\frac{\partial Sum_{nbOff}(t,x)}{\partial t} &= D\nabla^2 Sum_{nbOff}(t,x)\\ &\quad - k_d Sum_{nbOff}(t,x) + k_L R_{cr}(t,x)Sg_{nbOff}(t,x)\\ &\quad + k_L L_{cr}(t,x)Sg_{nbOff}(t,x) - k_T Sum_{nbOff}(t,x)\\ &\quad \times Th_{nbOff}(t,x)\end{aligned}\tag{8.36}$$

$$\begin{aligned}\frac{\partial Th_{nbOff}(t,x)}{\partial t} &= D\nabla^2 Th_{nbOff}(t,x) + 1.35k_p\\ &\quad - k_d Th_{nbOff}(t,x) - k_T Sum_{nbOff}(t,x)Th_{nbOff}(t,x)\end{aligned}\tag{8.37}$$

$$\begin{aligned}\frac{\partial Amp_{nbOff}(t,x)}{\partial t} &= D\nabla^2 Amp_{nbOff}(t,x) + k_p \\ &- k_d Amp_{nbOff}(t,x) - k_L Th_{nbOff}(t,x) Amp_{nbOff}(t,x)\end{aligned} \tag{8.38}$$

$$\begin{aligned}\frac{\partial nbrOff(t,x)}{\partial t} &= D\nabla^2 nbrOff(t,x) - k_d nbrOff(t,x) \\ &+ k_L Th_{nbOff}(t,x) Amp_{nbOff} y(t,x) \\ &- k_L nbrOff(t,x) Sg_{cr}(t,x)\end{aligned} \tag{8.39}$$

$$\begin{aligned}\frac{\partial Sg_{cr}(t,x)}{\partial t} &= D\nabla^2 Sg_{cr}(t,x) + 2k_p - k_d Sg_{cr}(t,x) \\ &- k_L On_{pr}(t,x) Sg_{cr}(t,x) - k_L nbrOff(t,x) Sg_{cr}(t,x)\end{aligned} \tag{8.40}$$

$$\begin{aligned}\frac{\partial Sum_{cr}(t,x)}{\partial t} &= D\nabla^2 Sum_{cr}(t,x) - k_d Sum_{cr}(t,x) \\ &+ k_L On_{pr}(t,x) Sg_{cr}(t,x) + k_L nbrOff(t,x) Sg_{cr}(t,x) \\ &- k_T Sum_{cr}(t,x) Th_{cr}(t,x)\end{aligned} \tag{8.41}$$

$$\begin{aligned}\frac{\partial Th_{cr}(t,x)}{\partial t} &= D\nabla^2 Th_{cr}(t,x) + 1.35k_p \\ &- k_d Th_{cr}(t,x) - k_T Sum_{cr}(t,x) Th_{cr}(t,x)\end{aligned} \tag{8.42}$$

$$\begin{aligned}\frac{\partial Amp_{cr}(t,x)}{\partial t} &= D\nabla^2 Amp_{cr}(t,x) + k_p \\ &- k_d Amp_{cr}(t,x) - k_L Sum_{cr}(t,x) Amp_{cr}(t,x)\end{aligned} \tag{8.43}$$

$$\begin{aligned}\frac{\partial Live(t,x)}{\partial t} &= D\nabla^2 Live(t,x) - k_d Live(t,x) \\ &+ k_L Sum_{cr}(t,x) Amp_{cr}(t,x) \\ &- k_L Live(t,x) Sg_{nx}(t,x)\end{aligned} \tag{8.44}$$

d. Boolean logic to determine if next state is *on*

$$\begin{aligned}\frac{\partial Sg_{nx}(t,x)}{\partial t} &= D\nabla^2 Sg_{nx}(t,x) + 2k_p \\ &- k_d Sg_{nx}(t,x) - k_L Live(t,x) Sg_{nx}(t,x) \\ &- k_L Br(t,x) Sg_{nx}(t,x)\end{aligned} \tag{8.45}$$

$$\begin{aligned}\frac{\partial Sum_{nx}(t,x)}{\partial t} &= D\nabla^2 Sum_{nx}(t,x) - k_d Sum_{nx}(t,x) \\ &+ k_L Live(t,x) Sg_{nx}(t,x) + k_L Br(t,x) Sg_{nx}(t,x) \\ &- k_T Sum_{nx}(t,x) Th_{nx}(t,x)\end{aligned} \tag{8.46}$$

$$\begin{aligned}\frac{\partial Th_{nx}(t,x)}{\partial t} &= D\nabla^2 Th_{nx}(t,x) + 0.65k_p \\ &- k_d Th_{nx}(t,x) - k_T Sum_{nx}(t,x) Th_{nx}(t,x)\end{aligned} \tag{8.47}$$

$$\begin{aligned}\frac{\partial Amp_{nx}(t,x)}{\partial t} &= D\nabla^2 Amp_{nx}(t,x) + k_p \\ &- k_d Amp_{nx}(t,x) - k_L Sum_{nx}(t,x) Amp_{nx}(t,x)\end{aligned} \tag{8.48}$$

$$\begin{aligned}\frac{\partial Nx(t,x)}{\partial t} &= D\nabla^2 Nx(t,x) - k_d Nx(t,x) \\ &+ k_L Sum_{nx}(t,x) Amp_{nx}(t,x)\end{aligned} \tag{8.49}$$

$$\begin{aligned}\frac{\partial On_{nx}(t,x)}{\partial t} &= D\nabla^2 On_{nx}(t,x) - k_d On_{nx}(t,x) \\ &+ k_d Nx(t,x) - k_L On_{nx}(t,x) Sg_{nxOn}(t,x)\end{aligned} \tag{8.50}$$

e. Boolean logic to determine if next state is *off*

$$\begin{aligned}\frac{\partial Nt_{nx}(t,x)}{\partial t} &= D\nabla^2 Nt_{nx}(t,x) - k_d Nt_{nx}(t,x) \\ &+ k_d Nx(t,x) - k_T Nt_{nx}(t,x) Off_{nx}(t,x)\end{aligned} \tag{8.51}$$

$$\begin{aligned}\frac{\partial Off_{nx}(t,x)}{\partial t} &= D\nabla^2 Off_{nx}(t,x) + k_p \\ &- k_d Off_{nx}(t,x) - k_T Nt_{nx}(t,x) Off_{nx}(t,x) \\ &- k_L Off_{nx}(t,x) Sg_{OffNx}(t,x)\end{aligned} \tag{8.52}$$

f. Clocked synchronization gates

$$\begin{aligned}\frac{\partial Sg_{nxOn}(t,x)}{\partial t} &= D\nabla^2 Sg_{nxOn}(t,x) + 2k_p \\ &- k_d Sg_{nxOn}(t,x) - k_L On_{nx}(t,x) Sg_{nxOn}(t,x) \\ &- k_L clk_{On}(t,x) Sg_{nxOn}(t,x)\end{aligned} \tag{8.53}$$

$$\begin{aligned}\frac{\partial On_{nx}Sum(t,x)}{\partial t} &= D\nabla^2 On_{nx}Sum(t,x) \\ &- k_d On_{nx}Sum(t,x) + k_L On_{nx}(t,x) Sg_{nxOn}(t,x) \\ &+ k_L clk_{On}(t,x) Sg_{nxOn}(t,x) \\ &- k_T On_{nx}Sum(t,x) On_{nx}Th(t,x)\end{aligned} \tag{8.54}$$

$$\begin{aligned}\frac{\partial On_{nx}Th(t,x)}{\partial t} &= D\nabla^2 On_{nx}Th(t,x) + 1.35k_p \\ &- k_d On_{nx}Th(t,x) - k_T On_{nx}Sum(t,x) On_{nx}Th(t,x)\end{aligned} \tag{8.55}$$

$$\begin{aligned}\frac{\partial On_{nx}Amp(t,x)}{\partial t} &= D\nabla^2 On_{nx}Amp(t,x) \\ &+ k_p - k_d On_{nx}Amp(t,x) - k_L On_{nx}Sum(t,x) \\ &\times On_{nx}Amp(t,x)\end{aligned} \tag{8.56}$$

$$\begin{aligned}\frac{\partial Set_{Bfr}(t,x)}{\partial t} &= D\nabla^2 Set_{Bfr}(t,x) - k_d Set_{Bfr}(t,x) \\ &+ k_L On_{nx}Sum(t,x) On_{nx}Amp(t,x)\end{aligned} \tag{8.57}$$

$$\begin{aligned}\frac{\partial Sg_{OffNx}(t,x)}{\partial t} &= D\nabla^2 Sg_{OffNx}(t,x) + 2k_p \\ &- k_d Sg_{OffNx}(t,x) - k_L Off_{nx}(t,x) Sg_{OffNx}(t,x) \\ &- k_L clk_{Off}(t,x) Sg_{OffNx}(t,x)\end{aligned} \tag{8.58}$$

$$\begin{aligned}\frac{\partial Sum_{OffNx}(t,x)}{\partial t} &= D\nabla^2 Sum_{OffNx}(t,x) \\ &- k_d Sum_{OffNx}(t,x) + k_L Off_{nx}(t,x) Sg_{OffNx}(t,x) \\ &+ k_L clk_{Off}(t,x) Sg_{OffNx}(t,x) \\ &- k_T Sum_{OffNx}(t,x) Th_{OffNx}(t,x)\end{aligned} \tag{8.59}$$

$$\frac{\partial Th_{OffNx}(t,x)}{\partial t} = D\nabla^2 Th_{OffNx}(t,x) + 1.35k_p - k_d Th_{OffNx}(t,x) - k_T Sum_{OffNx}(t,x) Th_{OffNx}(t,x) \tag{8.60}$$

$$\frac{\partial Amp_{OffNx}(t,x)}{\partial t} = D\nabla^2 Amp_{OffNx}(t,x) + k_p - k_d Amp_{OffNx}(t,x) - k_L Sum_{OffNx}(t,x) \times Amp_{OffNx}(t,x) \tag{8.61}$$

$$\frac{\partial Res_{Bfr}(t,x)}{\partial t} = D\nabla^2 Res_{Bfr}(t,x) - k_d Res_{Bfr}(t,x) + k_L Sum_{OffNx}(t,x) Amp_{OffNx}(t,x) \tag{8.62}$$

4. Storage stage

   a. Copies of Set/Res signals

$$\frac{\partial Set(t,x)}{\partial t} = D\nabla^2 Set(t,x) - k_d Set(t,x) + k_d Set_{Bfr}(t,x) - k_L Set(t,x) N1Sg(t,x) \tag{8.63}$$

$$\frac{\partial Res(t,x)}{\partial t} = D\nabla^2 Res(t,x) - k_d Res(t,x) + k_d Res_{Bfr}(t,x) - k_L Res(t,x) N2Sg(t,x) \tag{8.64}$$

   b. Flip-flop module

$$\frac{\partial ffBfrd(t,x)}{\partial t} = D\nabla^2 ffBfrd(t,x) - k_d ffBfrd(t,x) + k_L N2Th(t,x) N2(t,x) \tag{8.65}$$

$$\frac{\partial ffFbackNot(t,x)}{\partial t} = D\nabla^2 ffFbackNot(t,x) - k_d ffFbackNot(t,x) + k_L N1Th(t,x) N1(t,x) - k_L ffFbackNot(t,x) N2Sg(t,x) \tag{8.66}$$

$$\frac{\partial N1Th(t,x)}{\partial t} = D\nabla^2 N1Th(t,x) + 0.65k_p - k_d N1Th(t,x) - k_T N1Sum(t,x) N1Th(t,x) \tag{8.67}$$

$$\frac{\partial N2Sum(t,x)}{\partial t} = D\nabla^2 N2Sum(t,x) - k_d N2Sum(t,x) + k_L Res(t,x) N2Sg(t,x) + k_L ffFbackNot(t,x) N2Sg(t,x) - k_T N2Sum(t,x) N2Th(t,x) \tag{8.68}$$

$$\frac{\partial ffFback(t,x)}{\partial t} = D\nabla^2 ffFback(t,x) - k_d ffFback(t,x) + k_d ffBfrd(t,x) - k_L ffFback(t,x) N1Sg(t,x) \tag{8.69}$$

$$\frac{\partial N1Sg(t,x)}{\partial t} = D\nabla^2 N1Sg(t,x) + 2k_p - k_d N1Sg(t,x) - k_L Set(t,x) N1Sg(t,x) - k_L ffFback(t,x) N1Sg(t,x) \tag{8.70}$$

$$\frac{\partial N1Sum(t,x)}{\partial t} = D\nabla^2 N1Sum(t,x) - k_d N1Sum(t,x) + k_L Set(t,x) N1Sg(t,x) + k_L ffFback(t,x) N1Sg(t,x) - k_T N1Sum(t,x) N1Th(t,x) \tag{8.71}$$

$$\frac{\partial N1(t,x)}{\partial t} = D\nabla^2 N1(t,x) + k_p - k_d N1(t,x) - k_L N1Th(t,x) N1(t,x) \tag{8.72}$$

$$\frac{\partial N2Sg(t,x)}{\partial t} = D\nabla^2 N2Sg(t,x) + 2k_p - k_d N2Sg(t,x) - k_L Res(t,x) N2Sg(t,x) - k_L ffFbackNot(t,x) N2Sg(t,x) \tag{8.73}$$

$$\frac{\partial N2Th(t,x)}{\partial t} = D\nabla^2 N2Th(t,x) + 0.65k_p - k_d N2Th(t,x) - k_T N2Sum(t,x) N2Th(t,x) \tag{8.74}$$

$$\frac{\partial N2(t,x)}{\partial t} = D\nabla^2 N2(t,x) + k_p - k_d N2(t,x) - k_L N2Th(t,x) N2(t,x) \tag{8.75}$$

   c. Stored state

$$\frac{\partial Last(t,x)}{\partial t} = D\nabla^2 Last(t,x) - k_d Last(t,x) + k_d ffBfrd(t,x) \tag{8.76}$$

References

Allen PB, Chen X, Ellington AD (2012) Spatial control of DNA reaction networks by DNA sequence. Molecules 17: 13390–13402

Baker MD, Wolanin PM, Stock JB (2006) Signal transduction in bacterial chemotaxis. Bioessays 28(1):9–22

Bánsági T, Vanag VK, Epstein IR (2011) Tomography of reaction–diffusion microemulsions reveals three-dimensional Turing patterns. Science 331(6022):1309–1312

Chen Y, Dalchau N, Srinivas N, Phillips A, Cardelli L, Soloveichik D, Seelig G (2013) Programmable chemical controllers made from DNA. Nat Nanotechnol 8(10):755–762

Chirieleison SM, Allen PB, Simpson ZB, Ellington AD, Chen X (2013) Pattern transformation with DNA circuits. Nat Chem 5:1000–1005

Codd EF (1968) Cellular automata. Academic Press Inc, San Diego

Codon A, Kirkpatrick B, Maňuch J (2012) Reachability bounds for chemical reaction networks and strand displacement systems. DNA Computing and Molecular Programming. Springer, Heidelberg, Berlin

Cook M (2004) Universality in elementary cellular automata. Complex Syst 15(1):1–40

Dalchau N, Seelig G, Phillips A (2014) Computational design of reaction–diffusion patterns using DNA-based chemical reaction networks. DNA computing and molecular programming. Springer, Heidelberg, Berlin

Danino T, Mondragn-Palomino O, Tsimring L, Hasty J (2010) A synchronized quorum of genetic clocks. Nature 463(7279): 326–330

Doty D (2014) Timing in chemical reaction networks. In: Proceedings of the 25th ACM-SIAM symposium on discrete algorithms, pp 772–784

Du Y, Lo E, Ali S, Khademhosseini A (2008) Directed assembly of cell-laden microgels for fabrication of 3D tissue constructs. In; Proceedings of the National Academy of Sciences 105(28):9522–9527

Fujibayashi K, Hariadi R, Park SH, Winfree E, Murata S (2007) Toward reliable algorithmic self-assembly of DNA tiles: a fixed-width cellular automaton pattern. Nano Lett 8(7):1791–1797

Gács P (2001) Reliable cellular automata with self-organization. J Stat Phys 103(1/2):45–267

Gács P, Reif J (1988) A simple three-dimensional real-time reliable cellular array. J Comput Syst Sci 36(2):125–147

Lakin M, Phillips A, Stefanovic D (2013) Modular verification of DNA strand displacement networks via serializability analysis. DNA computing and molecular programming. Springer, Heidelberg, Berlin

Greenfield D, McEvoy AL, Shroff H, Crooks GE, Wingreen NS, Betzig E, Liphardt J (2009) Self-organization of the *Escherichia coli* chemotaxis network imaged with super-resolution light microscopy. PLoS Biol. 7(6)

Langton CG (1984) Self-reproduction in cellular automata. Phys D 10(1):135–144

Lindenmayer A (1968) Mathematical models for cellular interactions in development I. filaments with one-sided inputs. J Theor Biol 18(3):280–299

Lukacs G, Haggie P, Seksek O, Lechardeur D, Verkman NFA (2000) Size-dependent DNA mobility in cytoplasm and nucleus. J Biol Chem 275(1625)

Montagne K, Plasson R, Sakai Y, Fujii T, Rondelez Y (2011) Programming an *in vitro* DNA oscillator using a molecular networking strategy. Mol Sys Biol 7(1)

Murray JD (2003) Mathematical biology II: spatial models and biomedical applications, 3rd edn. Springer, New York

Neary T, Woods D (2006) P-completeness of cellular automaton rule 110. LNCS 4051(132–143)

Nehaniv CL (2004) Asynchronous automata networks can emulate any synchronous automata network. Int J Algebra Comput 14(05):719–739

von Neumann J, Burks AW (1966) The theory of self-reproducing automata. University of Illinois Press, Urbana

Qian L, Soloveichik D, Winfree E (2011) Efficient turing-universal computation with DNA polymers. DNA computing and molecular programming pp 123–140

Qian L, Winfree E (2011) Scaling up digital circuit computation with DNA strand displacement. Science 332(6034):1196–1201

Qian L, Winfree E (2011) A simple DNA gate motif for synthesizing large-scale circuits. J R Soc Interface 8(62):1281–1297

Qian L, Winfree E (2014) Parallel and scalable computation and spatial dynamics with DNA-based chemical reaction networks on a surface. DNA computing and molecular programming. Springer, Heidelberg, Berlin

Rothemund PWK, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol 2(12):e424

Ruiza SA, Chen CS (2007) Microcontact printing: a tool to pattern. Soft Matter 3:168–177

Sayama H (1999) A new structurally dissolvable self-reproducing loop evolving in a simple cellular automata space. Artif Life 5(4):343–365

Scalise D, Schulman R (2014) Designing modular reaction–diffusion programs for complex pattern formation. Technology 2(01):55–66

Seelig G, Soloveichik D, Zhang DY, Winfree E (2006) Enzyme-free nucleic acid logic circuits. Science 314:1585–1588

Smith DE, Perkins TT, Chu S (1996) Dynamical scaling of DNA diffusion coefficients. Macromolecules 29(4):1372–1373

Soloveichik D, Cook M, Winfree E, Bruck J (2008) Computation with finite stochastic chemical reaction networks. Nat Comput 7(4): 615–633

Soloveichik D, Seelig G, Winfree E (2010) DNA as a universal substrate for chemical kinetics. In: Proceedings of the National Academy of Sciences 107(12):5393–5398

Steinbock O, Kettunen P, Showalter K (1996) Chemical wave logic gates. J Phys Chem 100(49):18970–18975

Stellwagen E, Lu Y, Stellwagen N (2003) Unified description of electrophoresis and diffusion for DNA and other polyions. Biochemistry 42:11745

Tomita K, Kurokawa H, Murata S (2002) Graph automata: natural expression of self-reproduction. Phys D: Nonlin Phenom 171(4): 197–210

Tóth Ágota, Showalter K (1995) Logic gates in excitable media. J Chem Phys 103(6):2058–2066

Turing AM (1952) The chemical basis of morphogenesis. Phil T R Soc B 237:37–72

Wu A, Rosenfeld A (1979) Cellular graph automata. I. basic concepts, graph property measurement, closure properties. Inf Control 42(3):305–329

Zhang DY, Winfree E (2009) Control of DNA strand displacement kinetics using toehold exchange. J Am Chem Soc 131(47): 17303–17314