Journal of Statistical Physics, Vol. 114, Nos. 1/2, January 2004 (© 2004)

Computation by Asynchronously Updating Cellular Automata

Susumu Adachi,[1] Ferdinand Peper,[1] and Jia Lee[1]

Received September 26, 2002; accepted July 24, 2003

A known method to compute on an asynchronously updating cellular automaton is the simulation of a synchronous computing model on it. Such a scheme requires not only an increased number of cell states, but also the simulation of a global synchronization mechanism. Asynchronous systems tend to use synchronization only on a local scale—if they use it at all. Research on cellular automata that are truly asynchronous has been limited mostly to trivial phenomena, leaving issues such as computation unexplored. This paper presents an asynchronously updating cellular automaton that conducts computation without relying on a simulated global synchronization mechanism. The two-dimensional cellular automaton employs a Moore neighborhood and 85 totalistic transition rules describing the asynchronous interactions between the cells. Despite the probabilistic nature of asynchronous updating, the outcome of the dynamics is deterministic. This is achieved by simulating delay-insensitive circuits on it, a type of asynchronous circuit that is known for its robustness to variations in the timing of signals. We implement three primitive operators on the cellular automaton from which any arbitrary delay-insensitive circuit can be constructed and show how to connect the operators such that collisions of crossing signals are avoided.

KEY WORDS: Asynchronous cellular automata; Moore neighborhood; totalistic rule; delay-insensitive circuit; signal; module; universal computation.

1. INTRODUCTION

Cellular automata (CA) are discrete dynamical systems that have been used extensively for studying computation,[1–5] self-reproduction,[1, 6, 5] computing by physics,[7] crystalline computing,[8] nanocomputing,[9] polymers,[10] fluid

[1] Communications Research Laboratory, 588-2 Iwaoka, Iwaoka-cho, Nishi-ku, Kobe City, 651-2492 Japan; e-mail: sadachi@crl.go.jp

dynamics,[11] etc. Most cellular automata models update their cells in parallel at discrete time steps, but such a synchronous updating scheme, while easy to handle, has its limitations. In the context of modeling systems in nature that lack a global clock, the use of a synchronous updating scheme may cause artifacts,[12–14] such as false correlations between cells or spurious attractors. In many models, the behavior obtained with synchronous updating disappears when randomized (asynchronous) updating schemes, like those in Monte Carlo simulations, are adopted. Examples are simulations of the Prisoner's dilemma,[15] where the pattern of cooperators and defectors reduces to one of global defection,[12] or cellular automata, where information and perturbations propagate less efficiently through asynchronous updating,[16] and the dynamics and basins of attraction change substantially with the updating scheme.[13, 17, 18] Other examples include random Boolean networks,[19] where attractors and cycles disappear in favor of a single or a few global attractors,[20] though some rudimentary "pseudo-periodic" behavior can be found in asynchronous Boolean networks when searching for it by genetic algorithms.[14] We also mention lattice spin systems,[21] where the choice of the update scheme in the Monte Carlo method used is so crucial that it tends to be considered an integral part of the model,[22, 23] and coupled-map lattices (lattices with continuously valued states; ref. 24), where attractors tend to disappear in favor of a few global attractors when updating is done asynchronously.[25–31] Especially for models used for studying self-synchronization of systems, in which asynchronous elements adjust their timings to each other, like ants in colonies,[32, 33] the brain,[34] and bushfires,[35, 36] the use of synchronous updating is unlikely to lead to a deeper understanding of the underlying mechanisms.

Synchronous timing also has its limitations when designing and building computers. The alternative, computers based on asynchronous circuitry, has been gaining increasing attention. Among its advantages[37–40] are: (1) lower power consumption and less heat dissipation, as only those parts of an asynchronous circuit that are active need to draw power, (2) less wiring, as no global clock signal needs to be distributed, (3) faster average speed, as timing need not be tuned to the slowest part of a circuit, (4) less dependence on the physical conditions and implementations with regard to the correct operation of the circuit, (5) suitability for modular design techniques, as timing relationships are less important. Even though these advantages tend to be more pronounced at higher integration densities, asynchronous circuits have yet to take root, not only because of a lack of available design, testing, and manufacturing infrastructure, but also because in CMOS technology asynchronous designs actually perform worse with respect to power consumption, wiring requirements, and speed. One reason for this is the overhead caused by the signaling protocol required for making up for the lack of a global clock signal. Another reason is the additional hardware required to avoid unstable behavior in the circuits. These disadvantages may be less of an issue, however, in implementations based on different technologies,(41, 42) such as RSFQ superconducting technology,(43, 44) molecular electronics,(45) molecule cascades,(46) and quantum dots CA.(47, 48)

In order to efficiently realize asynchronously updating computing schemes by alternative technologies, it is important to investigate how the nondeterministic and probabilistic behavior associated with such schemes is avoided. The existence of asynchronous computers proves the possibility of deterministic behavior in asynchronously updated schemes, but it is unclear at this point how such schemes materialize on lower levels, i.e., how deterministic behavior can be extracted from the simple low-level asynchronous interactions usually found on nanometer scales. An important issue in this context is how a computational history can be obtained that is independent of the order of asynchronous updating.(49) Though the invariant history property is undecidable, as proven in ref. 49 for one-dimensional CA, this does not preclude the existence of schemes possessing it. One technique to construct such schemes is the simulation of a synchronous model by an asynchronous model: the synchronous part guarantees deterministic behavior, while the asynchronous part provides freedom of updating order. For example, in ref. 50, a network of synchronously updated threshold units (Hopfield network) is simulated on a network of the same type but without constraints on the updating scheme. This construction requires both an increased number of units and convergence time. Another example is the simulation of a synchronous CA by an asynchronous CA, which also causes overhead, but then in the number of states: a synchronous CA with n states is simulated by an asynchronous CA with $3n^2$ states in refs. 51, 49, and 52, $4n^2$ states in ref. 53, n^2+2n states in ref. 54, and $O(n\sqrt{n})$ states in ref. 55.[2] It is pointed out in ref. 41 that the above asynchronous CA lack some of the advantages over synchronous schemes with regard to physical implementations, as the simulation requires every cell to be continuously busy with synchronizing itself with the others, which would cause high power consumption and heat dissipation of each cell.

To find more efficient ways to compute deterministically by asynchronously updated systems, a more direct method is necessary, a method

[2] This overhead is actually not so bad, because to implement a synchronous CA by software or hardware, each cell requires memory for storing both its current state as well as its new state, a doubling of the memory space, which effectively squares the number of states.

that does not need to rely on the simulation of global synchronization, but rather conducts synchronization on a local level.

In this paper we propose a CA that operates in precisely this way. Though it computes deterministically, it lacks a simulated global synchronization mechanism and it requires only those cells to be active that conduct computations, a characteristic of asynchronous systems. This result is obtained by simulating on the CA so-called delay-insensitive circuits, a type of asynchronous circuit whose correctness of operation is insensitive to arbitrary finite delays of signals (see, e.g., refs. 37, 39, and 56). To simulate a delay-insensitive circuit on an asynchronously updated CA, we design configurations on the CA of cells set in appropriate states that behave like signals and like modules operating on signals. The configurations for the signals can propagate autonomously through the cellular space of the CA in the absence of obstacles. The configurations for the modules operating on these signals can change the directions of signals and increase or decrease them in number. Put together in appropriate ways, the configurations operating on signals can be used to form delay-insensitive circuits by which computations can be conducted. Unlike asynchronously updating CA based on the same principle,(41, 57) the proposed CA employs a Moore neighborhood, which implies that each cell interacts with its direct orthogonal and diagonal neighbors. The CA has cells that can be in six possible states and that undergo transitions in accordance with a set of 85 totalistic transition rules, a type of rule in which the outcome of a cell's state transition is determined by the number of the cell's neighbors in certain states.(58) Often used in the context of systems modeled by mean-field theory, CA with totalistic transition rules may be closer to physical systems than other CA.(25, 58)

The results in this paper may not only open the way to massively parallel computation models with improved physical realizability, but also to more realistic CA-based models of physical and biological phenomena that lack an external clock signal.

This paper is organized as follows. In Section 2, we define asynchronously updated CA, and in Section 3 we design signals and describe how and how fast they propagate on an asynchronously updated cellular space. The subject in Section 4 is delay-insensitive circuits. We introduce three primitive modules by which any arbitrary delay-insensitive circuit can be constructed. In Section 5, we implement delay-insensitive circuits on an asynchronously updated CA. We show how the primitive modules can be implemented on an asynchronously updated CA and how they can be connected in circuits without collisions of crossing signals. As examples, we design a delay-insensitive NAND gate and an S-module (a type of 1-bit memory). We finish with concluding remarks.

2. ASYNCHRONOUSLY UPDATING CELLULAR AUTOMATA

A CA is a system in which identical finite automata are arranged as cells in a regularly structured d-dimensional array (in this paper $d=2$), such that they are mutually connected to their neighbors.(1–3, 59) Each cell can be in one state, which is a member of a finite state set. The possible cell states allowed in this paper are 0, 1, 2, 3, 4, and 5. We denote these states by the symbols in Fig. 1. A cell with state 0, called a *quiescent cell*, is commonly used for the background. The state of a cell at time t is updated to a state at time $t+1$ in accordance with transition rules, which are defined as a function f that has the neighboring cells' states and the cell's own state as inputs. Formally,

Definition 2.1. A *deterministic cellular automaton* (or simply *cellular automaton*) is a system defined by $A=(Z^d, N, Q, f, q_0)$, where Z is the set of all integers such that Z^d represents a d-dimensional array of the cellular space ($d \geqslant 1$). N is called a *neighborhood index*, denoted by $(n_1, n_2,..., n_n)$, where each $n_i \in Z^d$ and $n_1=(0, 0,..., 0)$. Given a cell $a \in Z^d$, each cell in $\{(a+n_1),..., (a+n_n)\}$ is a neighborhood cell of a. Moreover, Q is a finite set of states ($Q \neq \emptyset$), whereas $f\colon Q^n \to Q$ is a mapping called the *local transition function*. $q_0 \in Q$ is the *quiescent* state satisfying $f(q_0,..., q_0)=q_0$. A *configuration* in A is a mapping $c\colon Z^d \to Q$, which assigns to each cell in A a certain state from Q.

There are two well-known models according to which cells interact with neighboring cells. The first is the von Neumann neighborhood, in which each cell interacts with four neighboring cells—those north, south, east, and west of it. The second is the Moore neighborhood, in which each cell interacts with eight neighboring cells, which, in addition to the von Neumann neighborhood, includes the diagonal cells. Formally,

Definition 2.2. Let $A=(Z^2, N, Q, f, q_0)$ be a two-dimensional CA. A is said to have a *von Neumann neighborhood* (see Fig. 2(a)) if

$$N=((0, 0), (0, -1), (1, 0), (0, 1), (-1, 0)).$$

symbol	□	▒	■	⊠	⊡	▦
state	0	1	2	3	4	5

Fig. 1. The symbols by which the cell states are encoded.

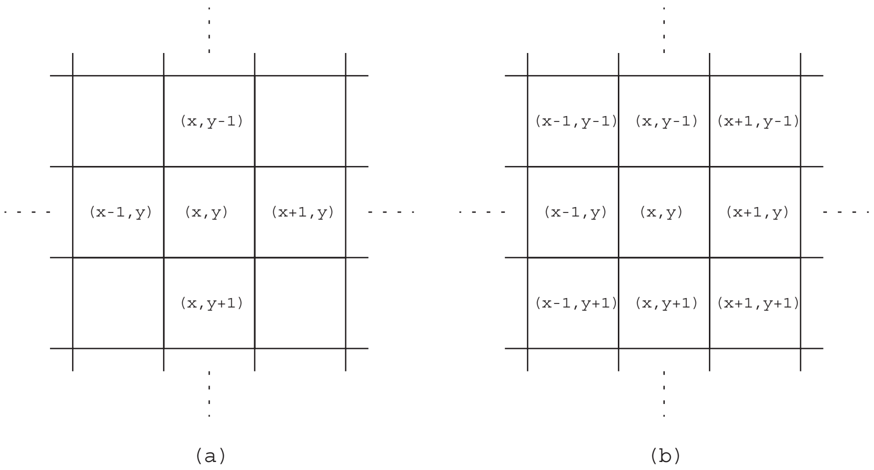


Fig. 2. Two-dimensional cellular spaces with (a) von Neumann neighborhood and (b) Moore neighborhood.

A is said to have a *Moore neighborhood* (see Fig. 2(b)) if

$$N = ((0,0), (0,-1), (1,-1), (1,0), (1,1), (0,1), (-1,1), (-1,0), (-1,-1)).$$

In CA research, mostly symmetric and totalistic transition rules have been the topic of interest. In a *symmetric* rule the pattern formed by the state of a cell and the states of the neighboring cells, including their reflection and rotation symmetric forms, determines the next state of the cell in a transition. In a *totalistic* rule the number of neighboring cells in a certain state, together with the state of the cell, determines the next state of the cell in a transition. As we will only use totalistic rules in this paper, we formally define only them:

Definition 2.3. Let $A = (Z^2, N, Q, f, q_0)$ be a two-dimensional CA with Moore neighborhood. A is *totalistic* (sometimes called *outer-totalistic* or *semi-totalistic*) iff $Q \subset Z$ and the local transition function f has the form

$$c'_0 = f(c_0, n_0, n_1, \ldots, n_{m-1}), \tag{1}$$

$$n_k = |\{c_i \mid c_i = k\}|, \tag{2}$$

where c_0 and c_i $(i = 1, 2, \ldots, 8)$ are the states of a cell and its eight neighbors respectively before the update, c'_0 is the state of the cell after the update (see Fig. 3), and the variable n_k denotes the number of cells in state k $(k = 0, 1, 2, \ldots, m-1)$, whereby m is the number of states.

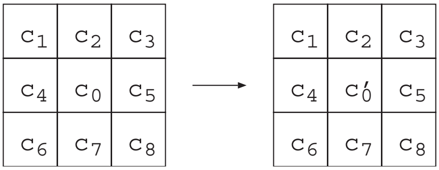


Fig. 3. Update scheme according to which a cell undergoes a transition.

A well-known (synchronously updated) CA with a Moore neighborhood and totalistic transition rules is the Game of Life.(59) This CA is based on a two-dimensional array of cells, each of which can be in one of two states, dead or alive. Though only a few transition rules are required, this model exhibits a remarkably rich behavior, which has been shown to be computationally universal.

Synchronously updated CA require all cells to undergo state transitions simultaneously at every time step. Consequently, the global states are uniquely determined at all times by the local transition function. If cell transitions occur in accordance with some other updating scheme, for example randomly and independently of each other,(16–18) the resulting model is an *asynchronously updated cellular automaton* (ACA). In this paper we limit ourselves to a scheme in which cells to be updated are randomly selected one by one. Formally,

Definition 2.4. Let $A=(Z^2, N, Q, f, q_0)$ be a CA. A is *asynchronously updating* if at each time step only one cell that is randomly selected from Z^2 undergoes a state transition, whereby each cell has a probability between 0 and 1 of being selected.

This updating method resembles Monte Carlo updating, a scheme used in, for example, the simulation of the relaxation of spin lattices, in which the transition probability of each cell depends on the difference of the energies before and after update in accordance with some algorithm, such as the heat-bath algorithm(60) (Gibbs sampler(61)) or the Metropolis algorithm.(62) In this paper we make no assumptions on the transition probabilities other than the assumptions in Definition 2.4, which guarantees that two neighboring cells are never updated simultaneously. This simplifies the design of the CA,(63) as compared to asynchronous CA that update their cells at arbitrary times independent of each other,(51, 57) which may occasionally result in neighboring cells being updated simultaneously. In an ACA, the global states are not necessarily uniquely determined at all times by the local transition function, because of the randomness by which transitions are timed.

In the formulations of the transition rules in this paper, we shall leave out rules that keep a cell's state unchanged. Such dummy transition rules are actually quite numerous, but we feel justified in ignoring them, since they do not contribute anything of value to our models. The updating scheme for an ACA may thus be interpreted as the updating scheme above, being applied under the condition that the states of a cell and its neighbors match the left-hand-side of at least one of the transition rules. A cell's state will remain unchanged as long as no such matches can be made when it is selected by the updating scheme to undergo a transition.

3. SIGNALS

To conduct non-trivial operations on CA, signals are required. A signal is a configuration of cells in certain states that moves in a certain direction over the cells as time passes. The basic form of a signal consists of one state-2 cell (the core) and three state-1 cells (the sheath) covering the part of the core opposite to the direction the signal moves into, which is diagonal. This sheath isolates the core from the rest of the cellular space to ensure that the core can only extend into the direction in which the sheath is missing. The signal has one of the two standard forms of signals in ACA identified in ref. 63. We call a continuous area of cells along which a signal propagates a *path*. For example, the signal in Fig. 4 propagates along a path in the southeastern direction, whereby its front part is extended step by step, while its tail is withdrawn by about the same pace.

The transition rules for signal propagation are given in Table I. Starting from a signal in its basic form in Fig. 4(a), we obtain a form in which the core is extended diagonally in the propagation direction due to rule 1, resulting in a state-3 cell in front of the original core (see Fig. 4(b)). This is followed by a forward extension of the sheath at the left or right side of the core due to rule 2, which may result in either form in Fig. 4(c), depending on which side is updated first. The other side of the sheath then follows due to rule 3, resulting in the form in Fig. 4(d). Rules 12 and 13 are included for technical reasons. Though they do not contribute to signal propagation—they actually slow down signals by making the sheath withdraw instead of extend—they are required to control interactions between signals in the vicinity of certain configurations that process signals, to be described in Section 5. In any case, the situation in Fig. 4(d) needs to occur before the process can progress any further. Subsequently, the cell at the back of the core is changed from state 2 to state 1 if there is one state-3 cell in front of it and five state-1 cells surrounding it, as in the transition from Fig. 4(d) to (e).

Table I. Transition Rules for the Propagation of a Signal. The Numbers in Column n_k Denote, from Left to Right, the Number of Neighboring Cells in State 0, State 1, State 2, etc.

No.	c_0	n_k	c'_0
1	0	701000	3
2	0	511100	1
3	0	421100	1
4	2	250100	1
5	3	530000	2
6	1	350000	0
7	1	440000	0
8	1	530000	0
9	1	620000	0
10	1	710000	0
11	1	800000	0
12	1	511100	0
13	1	421100	0

To erase the signal's tail, cells in state 1 that have only state-0 and state-1 cells as neighbors are reset to state 0, like in the transitions from Fig. 4(e) to the different possible configurations in Fig. 4(i). The erasure of the tail may also be postponed until the state-3 cell forming the head of the signal in Fig. 4(e) is transformed into a normal core, i.e., into a state-2 cell, as in Fig. 4(f). In any case, we will eventually find ourselves in situations resembling those in Fig. 4(j) or (k), which form the start of a new step forward of the signal. By repeating the above process, the signal is transmitted through the asynchronous cellular space.

The successive configurations in Fig. 4 are typical for signal propagation. Due to the asynchronicity of the model, the order in which the cells undergo transitions may vary, and even subsequences of transitions different from those in Fig. 4 may occur. The standard form of the signal in Fig. 4(a) and the lower part of Fig. 4(j) does not necessarily occur during propagation, because, due to the asynchronous character of the ACA, the withdrawal of the tail may be delayed when the core has already been extended forward, as in Fig. 4(k).

Figure 5 shows a sequence of signals propagating along the same path. Subsequent signals remain separated due to a regime in which a signal's tail should first be cleared before the signal behind it can be extended forward. In other words, a signal waits to extend its core until the remainder of the preceding signal in front of it is cleared. As a result, successive signals moving along the same path always remain separated by at least one quiescent cell.

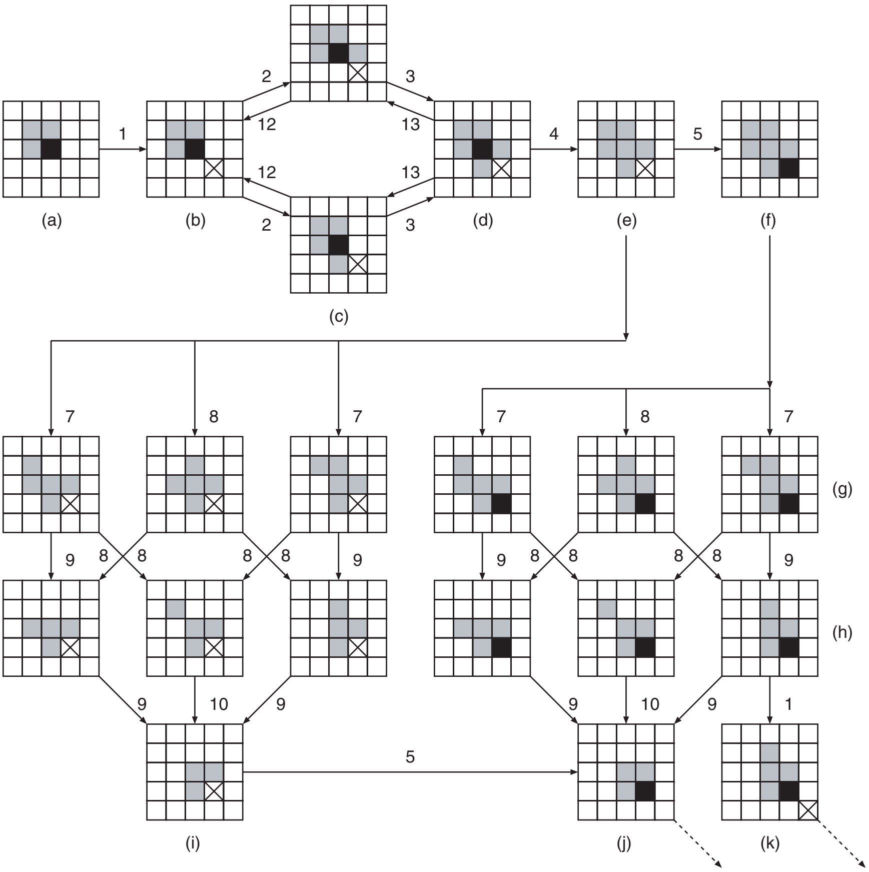


Fig. 4. Propagation of a signal on the asynchronously updated CA. The labels of the arrows between the subsequent configurations denote the rules in Table I according to which the transitions take place. Starting from the standard configuration of a signal in (a), the core is extended forward, followed by the sheath at the left and right sides of the signal, which brings us eventually to configuration (d). The withdrawal of the sheath back from (d) to (c) and (b) is an artifact of rules 12 and 13, which, though not strictly necessary for signal propagation, is used to smoothly process signals by some of the configurations in Section 5. From (d) on, the tail of the signal is withdrawn, which may be done in a variety of ways. Eventually, a configuration as in (j) will emerge, though some variations on this may be possible, like (k), as it cannot be guaranteed that the tail of the signal withdraws before the core extends forward again. Transitions like the one from the lower configuration in (i) to the lower configuration in (j) by rule 5 may also occur from the other configurations in (i) in rows (g) or (h) to the corresponding configurations in (j) in rows (g) or (h) due to rule 5. The arrows denoting these transitions are left out for reasons of simplicity.

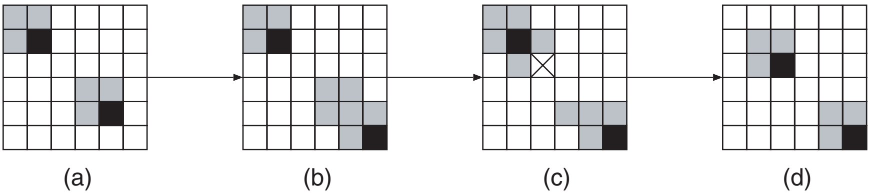


Fig. 5. Two subsequent signals on the ACA. The signal at the back will not extend its core forward unless the signal in front of it has withdrawn its tail. This guarantees that there is always a space between the signals of at least one quiescent cell.

The distance a signal travels on an ACA with 100×100 cells as a function of time averaged over 10^4 simulations is shown in Fig. 6, together with its standard deviation. The x-axis denotes the time steps t and the y-axis denotes the distance x of signal propagation. One time step corresponds to 10^4 times selecting one cell randomly and updating it. All cells in the ACA being equal in the probability of being selected for update, this results in one update on the average per time step of each of the cells. The average distance $\langle x \rangle$ traveled by a signal is 0.125 cells per step, and the variance of the distance is proportional to the number of steps (we measured $\langle \Delta x^2 \rangle \sim 0.06t$). The distribution of the distance converges to a Gaussian distribution as the number of steps increases.

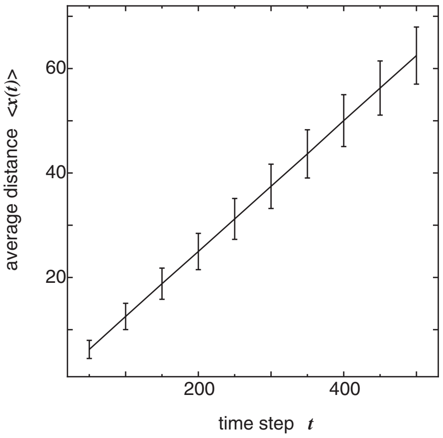


Fig. 6. Average distance of a signal propagating on an ACA with 100×100 cells as a function of time. Each cell is updated one time per step on the average, and this results in the signal advancing by approximately 0.125 cells per step on the average.

4. DELAY-INSENSITIVE CIRCUITS

A *Delay-Insensitive* (DI) circuit is an asynchronous circuit whose operation is robust to arbitrary signal delays. A DI-circuit needs no central clock since it is driven by input signals; it may thus be called an *event-driven* circuit. The circuit is composed of *path*s (wires) and *module*s. Signals are transmitted along the paths and are processed by the modules.

Any arbitrary Boolean circuit in a synchronous system can be constructed from a fixed set of primitive operators, for example from AND-gates and NOT-gates. Unfortunately, such Boolean gates are not suitable as primitives for DI circuits because they lack the functionality needed to cope with delay-insensitivity.(64, 65) Primitives that do form a fixed set from which any arbitrary DI circuit can be constructed—called a *universal set*—have been the subject of a few proposals.(39, 56, 42) We use a universal set of primitives based on those in ref. 42 that is particularly suited for the ACA in this paper. Unlike the primitive modules for DI circuits with up to five or six paths proposed in refs. 39 and 56, the primitive modules we use have four or less bi-directional paths, i.e., paths that can be used for both input and output, albeit not at the same time. Moreover, a path can simultaneously contain multiple signals, as in Fig. 5.

We define the following three modules as DI circuit primitives:

– *FORK.* A module with one input path and two output paths, as shown in Fig. 7(a). Upon receiving an input signal, it produces one output signal on each of its output paths.

– *S-JOIN (Symmetric JOIN).* A module with three bi-directional paths, as shown in Fig. 7(b), each expressed by a double-headed arrow. This module is symmetric, in the sense that when it receives input signals from two of the paths, it sends an output signal along the remaining path. When it receives only one input signal, the signal is held until a signal is received from one of the other two paths. A signal that is held is called *pending*.

– *4-MERGE (4-Way MERGE).* A module with four bi-directional paths, as shown in Fig. 7(c), which has four functions. When it receives a signal from A or B, it sends an output signal to C. This functionality is called MERGE. When it receives signals from A and B simultaneously, it sends two successive signals to C. This functionality is called Parallel-MERGE (P-MERGE). When it receives a signal from C, it sends an output signal to D. This functionality, called Input/Output Multiplexing (IOM), is used together with the MERGE functionality to convert the bi-directional path C into one input path (A or B) and one output path (D).

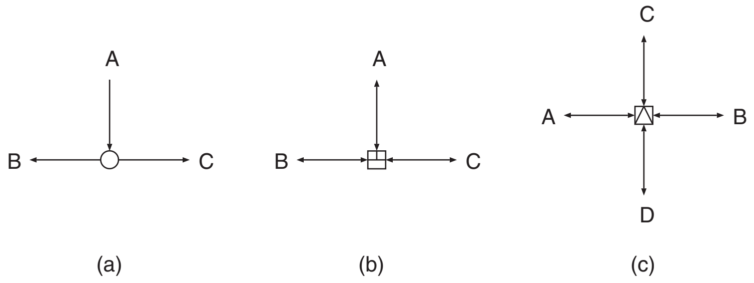


Fig. 7. Primitive modules: (a) a FORK module, (b) an S-JOIN module, and (c) a 4-MERGE module.

Finally, when the 4-MERGE module receives a signal from D, it waits until it receives a signal from A and then sends an output signal to B. The roles of A and B may also be reversed in this case. This functionality, called Arbitrating Test and Set (ATS), is used to arbitrate the access to a shared resource by competing processes, such as, for example, two signals that wish to pass through the same area at the same time (see crossing of signals in the next section).

This set of modules is universal, that is, any arbitrary DI circuit can be constructed from them. As an example, a module called TRIA,[39] constructed from one S-JOIN and three 4-MERGE modules, is shown in Fig. 8 (see also ref. 42). A TRIA module has three input and three output paths. When it receives signals from a (resp. b or c) and b (resp. c or a), it sends an output signal to p (resp. q or r).

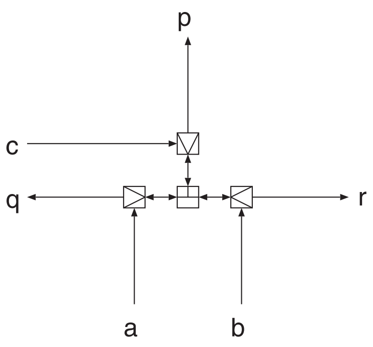


Fig. 8. A TRIA module constructed from one S-JOIN and three 4-MERGE modules.

5. IMPLEMENTING DELAY-INSENSITIVE CIRCUITS ON ASYNCHRONOUS CELLULAR AUTOMATA

5.1. Implementing the Primitives

To implement the three primitive modules {FORK, S-JOIN, 4-MERGE} in Section 4 on the proposed ACA, we use six states, of which states 4 and 5 are specifically for the modules.

The FORK module is composed of three cells in state 4, and it uses the transition rules in Table II in addition to some of the rules in Table I. The configurations of the FORK and its operations are shown in Fig. 9. The state-4 cell of the FORK nearest to the signal in Fig. 9(a) induces the signal to extend its core into a state-2 cell, as in Fig. 9(b). As this cell can be interpreted as a signal's core from two sides, it is further extended into a state-3 cell at each of its two sides, as in Fig. 9(c). The next step in the process is surrounding the two signals-to-be with sheath at appropriate places. This will eventually result in the configuration in Fig. 9(g), possibly via the intermediate configuration in Fig. 9(e). It may also result in the configurations in Fig. 9(d) or Fig. 9(f), due to the undesired (but possible) application of rules 2 and 3, respectively. While these configurations are dead ends, they cannot be avoided with absolute certainty, because rules 2 and 3 are required for signal propagation, as is shown in Section 3. The way out of this is provided by rules 12 and 13. Though they slow down the propagation of a signal, they also bring us back on the right track toward

Table II. Transition Rules Used for the FORK Module (Some of the Rules in Table I Are Also Used for the FORK)

No.	c_0	n_k	c'_0
14	0	601010	2
15	0	412100	1
16	0	401120	1
17	0	322100	1
18	0	232100	1
19	0	311120	1
20	0	221120	1
21	2	041210	3
22	3	520100	2
23	3	052010	1
24	1	520010	0
25	1	510020	0
26	1	600020	0
27	1	700010	0

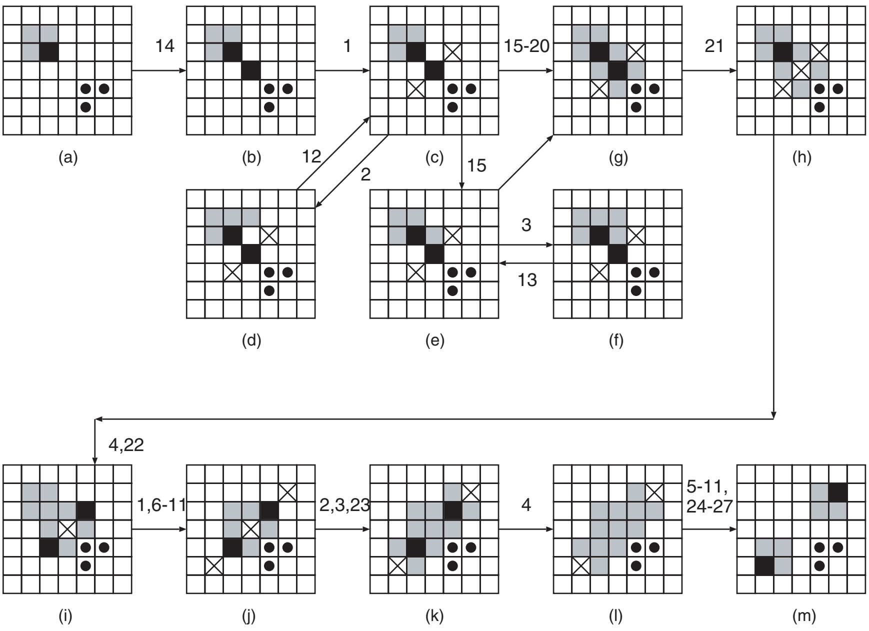


Fig. 9. Configuration and operation of the FORK module. The input signal triggers the formation of two output signals, after which it is destroyed. The dead-end configurations in (d) and (f) are artifacts of rules 2 and 3. Escape from them is possible due to rules 12 and 13, respectively.

the configuration in Fig. 9(g), as their effects are opposite to those of rules 2 and 3, respectively. The randomized nature of the update process means that at some time these rules will be applied, enabling our escape via the only way out of the dead ends. Next, after the two signals to be output have gathered sheath, they develop further along the lines of Fig. 9(h) to (m), in the process splitting up, and having the remainders of the input signal destroyed.

The S-JOIN module is composed of one cell in state 5, and it uses the transition rules in Table III, in addition to some of the rules in Tables I and II. The S-JOIN's configurations and operations are shown in Fig. 10(A) for signals arriving from opposite paths and in Fig. 10(B) for signals arriving from adjacent paths. A single signal input to the S-JOIN remains pending until a second signal arrives.

When two signals arrive from opposite paths, the state-5 cell of the S-JOIN induces the formation of a new state-2 cell, in turn giving rise to the formation of a state-3 cell (see Fig. 10(A-b)), which forms the front part of the output signal-to-be. Next, the sheath is formed, as in Fig. 10(A-c),

Table III. Transition Rules Used for the S-JOIN Module (Some of the Rules in Tables I and II Are Also Used for the S-JOIN)

No.	c_0	n_k	c'_0
28	0	502001	2
29	0	322001	1
30	0	232001	1
31	2	042101	3
32	3	061001	1
33	1	520001	0
34	1	610001	0
35	1	700001	0
36	1	501101	0
37	1	502001	0
38	0	233000	1
39	0	321101	1
40	0	143000	1
41	1	411101	0

after which the output signal develops further (Fig. 10(A-d)) and the input signals are destroyed, resulting in the situation in Fig. 10(A-e). In the subsequent stages the output signal develops further, clears the remainders of its tail, and breaks away from the S-JOIN (Fig. 10(A-j)).

When two signals arrive from adjacent paths to the S-JOIN, a more complicated situation arises. As signals are now directly next to each other (see Fig. 10(B-b)) an unintended interaction due to rule 3 may occur, resulting in the dead-end configuration in Fig. 10(B-c). The way to resolve this situation is similar to the way used with the FORK, i.e., by applying rule 13. Eventually, the configuration in Fig. 10(B-d) emerges, after which the operation continues along the same lines as with input signals arriving from opposite paths, i.e., by destroying the input signals, clearing the remainders of the output signal's tail, and breaking the output signal away from the S-JOIN, eventually resulting in a configuration as in Fig. 10(B-l).

The 4-MERGE module is composed of cells in states 4 and 5 and it uses the transition rules in Table IV, in addition to some of the transition rules in Tables I–III. The configurations and operations of the 4-MERGE are shown in Fig. 11.

The MERGE-functionality, illustrated by the sequence of configurations in Fig. 11(A), is mainly based on rules 42 to 61. The operation progresses along the same lines as with the FORK and the S-JOIN, with an output signal created by the MERGE, after which the input signal is destroyed. Again a dead-end situation (Fig. 11(A-f′)) occurs that resembles

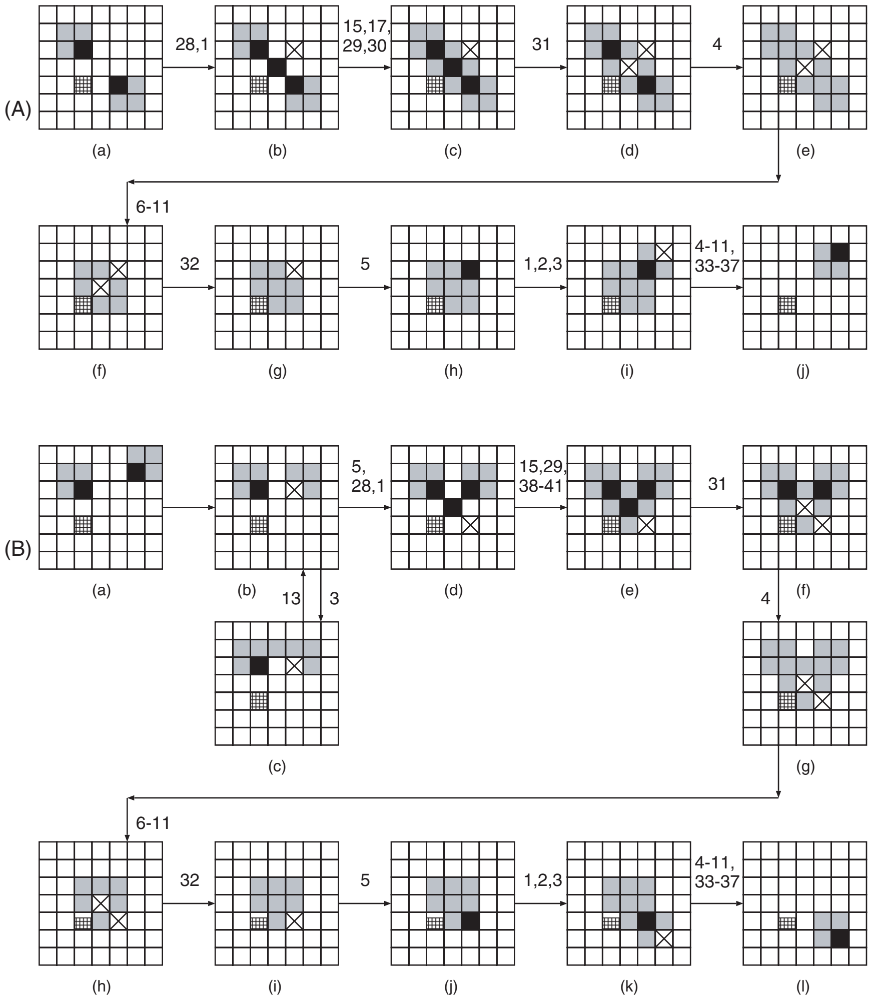


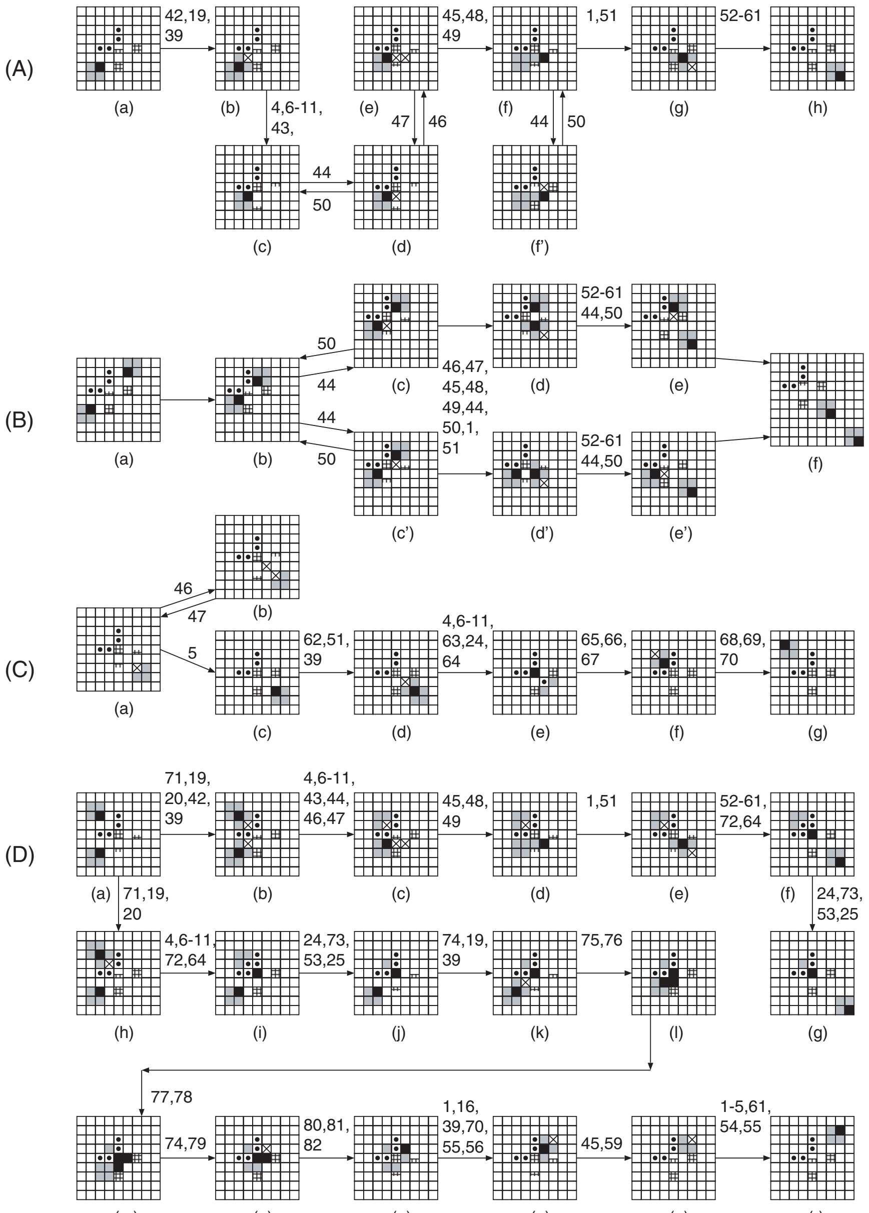
Fig. 10. Configuration and operation of the S-JOIN module. (A) Two signals input at opposite paths and (B) two signals input at adjacent paths. The two input signals trigger the formation of an output signal, after which the input signals are destroyed. The dead end (B-c) is an artifact of rule 3. Escape from it is only possible by the application of rule 13.

the situation we encountered before with rules 2 and 12 or rules 3 and 13. Here rule 44 is the culprit. While rule 44 is necessary for the transition from Fig. 11(A-c) to (A-d), it also causes a dead end. Rule 50 resolves this situation, at the price of slowing down the processing of the signal in the configuration in Fig. 11(A-d). A similar situation occurs with rules 46 and 47, as shown in Fig. 11(A-d) and (A-e). Rule 47 does not play a positive

Table IV. Transition Rules for the 4-MERGE Module (Some of the Rules in Tables I–III Are Also Used for the 4-MERGE)

No.	c_0	n_k	c'_0	No.	c_0	n_k	c'_0
42	0	301022	3	64	5	500030	2
43	3	130022	2	65	0	301040	3
44	0	311012	3	66	4	321002	0
45	2	030122	1	67	2	500120	5
46	0	400103	3	68	3	300041	2
47	3	400103	0	69	2	020141	1
48	3	220112	1	70	1	300041	0
49	3	410003	2	71	0	201041	3
50	3	311012	0	72	3	030041	4
51	0	411101	1	73	4	121040	1
52	2	130103	1	74	0	302021	3
53	1	420020	0	75	3	131021	2
54	1	430001	0	76	0	312011	2
55	1	310022	0	77	2	032021	1
56	1	410012	0	78	0	402011	2
57	1	500003	0	79	2	222011	1
58	1	500012	0	80	2	131120	5
59	1	320012	0	81	2	310112	1
60	1	230003	0	82	3	310022	2
61	1	240020	0	83	1	400022	0
62	0	401003	3	84	1	301022	0
63	3	230003	4	85	1	601001	0

Fig. 11. [See page 279.] Configuration and operation of the 4-MERGE module. (A) MERGE functionality. An input signal arrives from the lower left input path, giving rise to an output signal on the lower right path. Alternatively, when an input signal arrives from the upper left input path, the same happens. The dead end in (A-f′) is an artifact of rule 44. Escape from it is possible by rule 50, which has the opposite effect of rule 44. (B) P-MERGE functionality. One of the input signals is accepted first, whereas the other signal is kept pending until the first signal's processing is finished. (C) IOM functionality. The input signal arriving via the bi-directional path is redirected as output. The dead end in (C-b), caused by rule 46, is resolved by rule 47. (D) ATS functionality. Arbitration decides which of the two input signals enters first. If it is the lower input signal, the track from (D-b) to (D-g) is followed, resulting in a signal being output and the upper input signal being kept pending. The reverse transition of rule 44 between (D-b) and (D-c) is left out to save space in the figure. If the arbitration results in the upper signal entering first, the track from (D-h) to (D-r) is followed, resulting in one signal being output via the upper right path and no signal pending.


(A)
42,19, 39
4,6-11, 43,
44
50
47
46
45,48, 49
1,51
44
50
52-61
(a) (b) (c) (d) (e) (f) (f') (g) (h)
(B)
50
44
44
50
46,47, 45,48, 49,44, 50,1, 51
52-61 44,50
52-61 44,50
(a) (b) (c) (d) (e) (f) (c') (d') (e')
(C)
46
47
5
62,51, 39
4,6-11, 63,24, 64
65,66, 67
68,69, 70
(a) (b) (c) (d) (e) (f) (g)
(D)
71,19, 20,42, 39
4,6-11, 43,44, 46,47
45,48, 49
1,51
52-61, 72,64
71,19, 20
24,73, 53,25
4,6-11, 72,64
24,73, 53,25
74,19, 39
75,76
77,78
74,79
80,81, 82
1,16, 39,70, 55,56
45,59
1-5,61, 54,55
(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m) (n) (o) (p) (q) (r)

role in the MERGE functionality but is necessary to avoid the dead-end situation in the IOM functionality in Fig. 11(C-b).

The P-MERGE functionality, illustrated by the sequence of configurations in Fig. 11(B), is based on the same rules as the MERGE functionality. The main difference is that in the configuration in Fig. 11(B-b) arbitration is necessary to decide which of the two input signals is to be processed first. The two possible continuations, i.e., from (B-c) to (B-e) or from (B-c′) to (B-e′) respectively, go along the same lines as the MERGE. After the first input signal enters the P-MERGE, the second input signal is kept pending until the P-MERGE finishes processing of the first input signal. Eventually, for both continuations we wind up in a configuration resembling the one in Fig. 11(B-f) .

The IOM functionality, illustrated by the sequence of configurations in Fig. 11(C), is mainly based on rules 62 to 70. The operation progresses along the same lines as with previous configurations, with an output signal created by the IOM, after which the input signal is destroyed.

The ATS functionality, illustrated by the sequence of configurations in Fig. 11(D), is mainly based on rules 71 to 85. As with the P-MERGE, again we encounter arbitration. In the upper track, starting with the configuration in Fig. 11(D-b), the lower input signal is admitted first. The processing of the lower input signal is essentially similar to the processing according to the MERGE functionality, and it gives rise to one output signal. The upper input signal is kept in a pending state (Fig. 11(D-g)) that will last until the next input signal from the lower path arrives.

In the lower track, starting with the configuration in Fig. 11(D-h), the upper input signal is admitted first, after which it is made pending in Fig. 11(D-j). This pending state is resolved when the lower input signal enters the configuration. Eventually, the whole 4-MERGE is cleared of signals and a signal is output, as in Fig. 11(D-r). The arbitration encountered with the ATS functionality results in the divergence into the final two situations, (D-g) and (D-r), a situation different from that with the arbitration of the P-MERGE in Fig. 11(B). While this behavior is nondeterministic, it may still result in determinism on a more global, say circuit, scale, as we will see with the circuit for crossing signals below.

5.2. Combining the Primitives into Circuits

Realizing a particular delay-insensitive circuit is a matter of laying out the primitive modules on the CA, whereby output paths and input paths connected to each other are lined up on the cellular space such that signals flow from the corresponding module outputs to the corresponding module inputs. Special care with regard to this is required for paths that cross each other, since the CA lacks a third dimension via which crossing can be made. To prevent the collision of signals on crossing paths, the signals need to resolve which of them may pass first. A well-designed circuit will in most cases contain crossings over which only one signal needs to pass at a time. Such crossings are called *collision-free*. Even after careful design, however, there may still be crossings where this is not guaranteed. Though it is possible to design signals such that they can cross each other by adding transition rules, as in ref. 63, there is a more elegant way based on a module that is specifically designed for this task.

Such a crossing module is realized using only our primitives and only collision-free path crossings.[42] A circuit scheme of this module is given in Fig. 12(a), and its implementation on the ACA in Fig. 12(b). This module arbitrates between signals competing for a shared resource, which is the area where the signals cross. The arbitration conducted by the module is based on the ATS-functionality of the two 4-MERGE modules in the circuit. The crossing module can be used under all circumstances: the order by which the signals arrive at its inputs is irrelevant for its correct functioning: eventually all input signals pass the crossing.

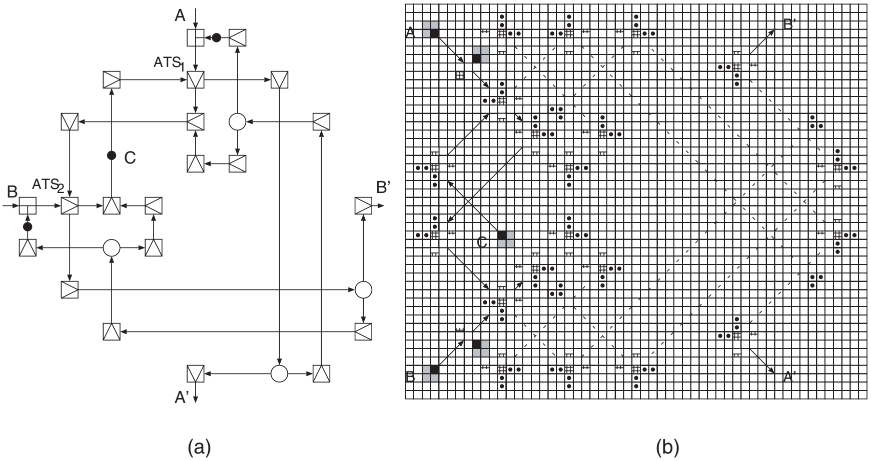


Fig. 12. (a) Delay-insensitive circuit of a crossing module and (b) its implementation on the ACA. The two 4-MERGE modules with ATS functionality, i.e., ATS_1 and ATS_2, record the arrival of signals from inputs A and B, respectively. Signal C loops around along the arrows to check the states of ATS_1 and ATS_2 alternatingly. If the state of ATS_1 (resp. ATS_2) indicates arrival of an input signal, the state is reset and an output signal is produced at A′ (resp. B′).

5.3. A Delay-Insensitive NAND Gate

The NAND gate, commonly used in synchronous circuits, has a delay-insensitive counterpart that conducts the logical NAND operation even if input signals are delayed. To represent a binary signal in a DI circuit, two paths are usually used, one labeled 0 and one labeled 1. Called *dual-rail encoding* (e.g., see ref. 40), this method encodes a 0 by a signal on the path labeled 0 and a 1 by a signal on the path labeled 1. Signals on both paths at the same time are forbidden, and the absence of a signal on either path indicates that no information is being transmitted. Using dual-rail encoding for its input and output paths, the NAND-gate requires four input paths, i.e., a_0 and a_1 encoding one input signal and b_0 and b_1 encoding the other input signal, as well as two output paths, encoded by c_0 and c_1.

The design of the delay-insensitive NAND gate is shown in Fig. 13(a).[42] It operates as follows.

– If a signal is input to each of the two paths in one of the sets $\{a_0, b_0\}$, $\{a_0, b_1\}$, or $\{a_1, b_0\}$, a signal is output to c_1.

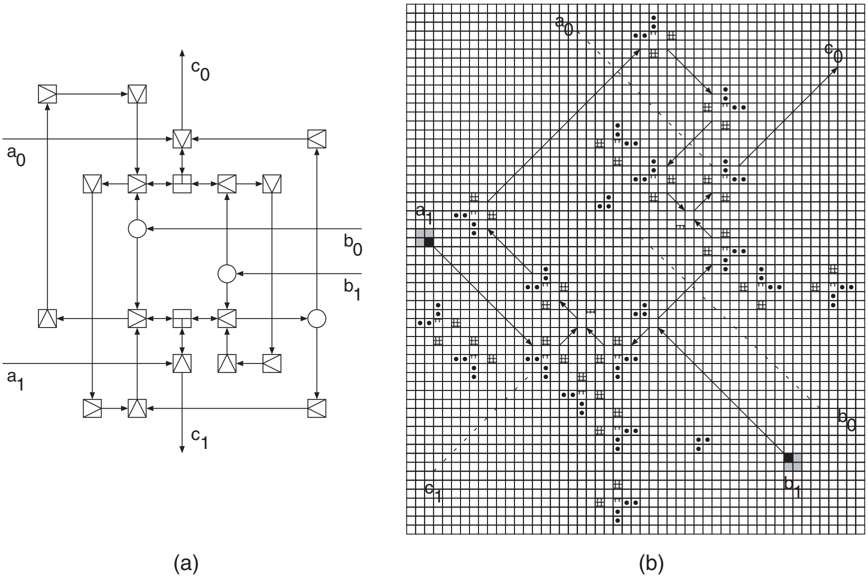


Fig. 13. (a) A delay-insensitive dual-rail encoded NAND gate and (b) its implementation on the proposed ACA. Paths a_0 and a_1 correspond to one input line in a conventional NAND-gate, paths b_0 and b_1 to the other input line, and paths c_0 and c_1 correspond to the output line.

– If a signal is input to each of the paths a_1 and b_1, a signal is output to c_0.

– Signals input to any other combinations of paths are illegal.

If there is only one input signal, it becomes pending, and the gate will wait for the second input signal. The implementation of the delay-insensitive NAND on the proposed ACA is shown in Fig. 13(b). In this figure, signals are input to the wires a_1 and b_1. As all crossings are collision-free, the crossing module of Fig. 12 is not required.

The dual-rail encoding method employed is very convenient, since it allows the construction of a NOT gate by just crossing two wires, reversing the outputs 0 and 1. It follows that a delay-insensitive AND gate can be constructed from the delay-insensitive NAND gate by simply reversing outputs c_0 and c_1, and a delay-insensitive OR gate can be constructed by reversing inputs a_0 and a_1, as well as reversing inputs b_0 and b_1 of the delay-insensitive NAND gate.

5.4. A 1-Bit Memory

The S-module (Select module) is a 1-bit memory that was first proposed in ref. 56. This module has three input paths, i.e., S, R, and T, and four output paths, i.e., S', R', T_0, and T_1. Its internal state can be 0 or 1. This module functions as follows:

– If a signal is input to S, the memory's state is set to 1, and an output signal is produced at S'.

– If a signal is input to R, the memory's state is set to 0, and an output signal is produced at R'.

– If a signal is input at T, an output signal is produced at T_i, where i is the state of the memory ($i = 0$ or 1).

We use a circuit design[(41)] for this module that requires only three TRIA modules, as shown in Fig. 14(a). The black blobs attached to two of the S-JOIN modules in the circuit scheme are signals that store the state of the memory. Each of these signals keeps pending at its S-JOIN in the absence of an input signal to the S-module.

The implementation of an S-module on the ACA is shown in Fig. 14(b). In this figure, the memory is in state 1, and it receives an input from its S-path. As all crossings are collision-free, the crossing module is not required.

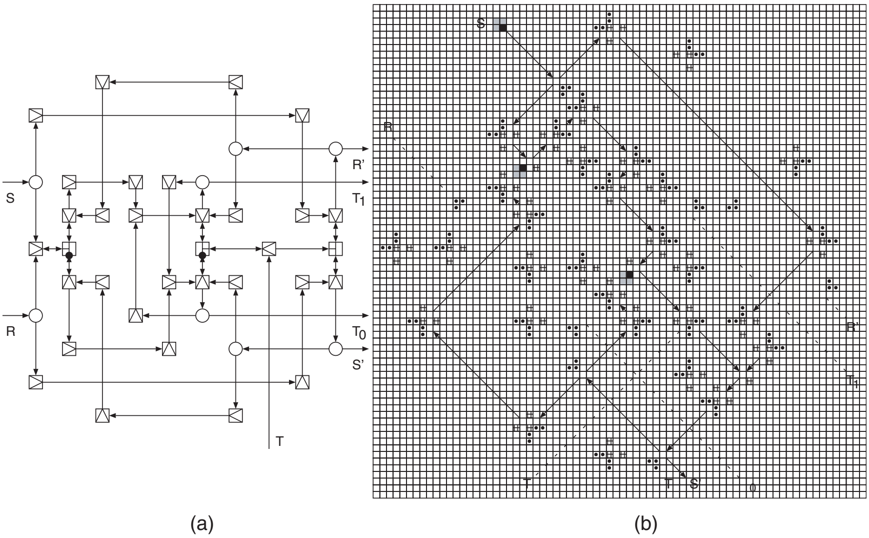


Fig. 14. (a) Delay-insensitive circuit of a 1-bit memory (S-module) and (b) its implementation on the ACA. The state of the memory is stored by two signals, indicated by black blobs, each of which is pending at an S-JOIN.

6. CONCLUSIONS AND DISCUSSION

This paper shows how to conduct deterministic computations on asynchronously updating cellular automata with a Moore neighborhood and totalistic transition rules. In our approach, synchronization is done locally, as opposed to the traditional approach by which a timing mechanism is simulated on the cellular automaton that keeps the cells approximately in pace with each other. The traditional approach is essentially a global synchronization scheme, since each of the cells is continuously busy to support synchronization, even if it conducts no computation. Our approach, on the other hand, requires only those cells on an asynchronously updating cellular automaton to be active that are within a few cells distance of a signal or a configuration processing a signal. We achieve this result by simulating delay-insensitive circuits on asynchronously updating cellular automata. Such circuits can be built from a few primitive modules, which are shown in ref. 42 to be sufficient for constructing an infinite circuit implementing a universal Turing Machine. This implies that the proposed asynchronously updated cellular automaton is computationally universal.

The delay-insensitive circuits employed in this paper are different from traditional delay-insensitive circuits in the sense that (1) they allow more than one signal on an interconnection line and (2) they allow bi-directional interconnection lines, i.e., lines on which signals can propagate in both directions, albeit never at the same time. Whereas these properties make our circuits hard to efficiently implement by solid-state electronics, they result in fewer interconnection lines for primitive modules,(42) making them suitable for cellular automata. Cellular automata allow multiple signals on a path in a natural way, only imposing the condition that subsequent signals should be separated by at least one quiescent cell, as in Fig. 5. Bi-directionality of paths follows from the rotational symmetry of the cellular space.

The correct operation of the delay-insensitive circuits in this paper depends on the correctness of the circuit design—a situation similar to that with other types of circuit. If no care is taken as to how primitive modules are combined and connected to each other, the resulting circuits may contain signals piling up on a path behind a signal that remains pending to a module, or they may contain signals coming from opposite directions on a bi-directional path that run into each other, etc. Such situations, also denoted as *deadlock*, are often characterized by different parts of a system waiting indefinitely for input from each other. For example, two S-JOIN primitives may mutually have to wait for input from each other, a situation that can never be resolved. Deadlock of this type is hard to prevent, because it is due to incorrect circuit design. Provided that circuits are correctly designed and laid out on the cellular automaton, however, deadlock will not occur, because an undefined combination of cell states will never arise.

The transition rules allow the cellular automaton to run correctly under an asynchronous timing scheme that randomly selects one of the cells for update in each time step. Do the interactions between the cells in an asynchronously updating cellular automaton differ in a principal way from interactions in synchronous cellular automata? To address this question we first note that the transition rules need to be designed without any order in mind by which cells are selected for update. Some control on this order is possible by using configurations in which each cell and its neighbors are in a unique combination of states, such that only one (or a few) transition rule(s) apply at a time. While this strategy tends to result in cellular automaton designs in which most cells change their states in a strictly sequential order (on a local scale), it leaves open situations in which a transition rule not only has the intended effect in some configurations, but where it also results in artifacts in other configurations. The way out of this is to add transition rules that undo these artifacts. This comes at a price, however. The transition rules added to undo artifacts may also undo the intended effects, thereby interfering with the operation of the cellular automaton. A striking example of this is the use of rule 2 for signal propagation (see Fig. 4), and its side effect in the FORK configuration, causing the dead end configuration in Fig. 9(d). Rule 12 can be used to escape from the dead-end, as it has exactly the opposite effect of rule 2. Unfortunately, in turn rule 12 too has a side effect, as it may apply to configurations for which it is not intended, reversing the result of rule 2 (Fig. 4(c)). The latter side effect is not lethal, however, because, due to the asynchronous nature of updating, there is a non-zero probability that not rule 12, but another rule (in this case rule 3, resulting in the configuration in Fig. 4(d)), is applied to the configuration, after which the sequence of transitions continues. Rule 12 thus does not block the propagation of the signal in Fig. 4, though it slows it down on the average. Similar effects occur with rules 3 and 13 being each other's opposites, rules 44 and 50, and rules 46 and 47. Such situations may have their counterparts in systems in nature, which may be characterized by interactions that are each others' reverse. An example of such systems may be non-processive (cooperative or single-headed) molecular motors (see ref. 66 for a review of Brownian motors), which, according to some theories on muscular contraction (e.g., ref. 67) employ both forward and backward movements of (myosin) heads to (actin) binding sites due to a loose mechanochemical coupling—an issue attracting much debate.(68) Interpreted in the light of our results, these forward–backward interactions may prevent a system from becoming stuck in spurious attractor states, while still being able to bias a process in a certain direction, albeit at a reduced speed.

As an asynchronous mode of operation is often found in nature, where, for example, chemical reactions only occur when the right molecules are available in the right positions, our approach promises improved ways to simulate certain kinds of physical systems by cellular automata.

ACKNOWLEDGMENTS

We thank Shinro Mashiko at the Communications Research Laboratory, Japan, for his support. We also thank the two anonymous reviewers for their valuable comments. This research is financed by the Ministry of Public Management and Home Affairs in Japan.

REFERENCES

1. J. von Neumann, *Theory of Self-Reproducing Automata* (University of Illinois Press, 1966).
2. E. F. Codd, *Cellular Automata* (Academic Press, New York, 1968).
3. E. R. Banks, Universality in cellular automata, *IEEE 11th Ann. Symp. Switching and Automata Theory* (1970), pp. 194–215.

4. T. Serizawa, Three-state Neumann neighbor cellular automata capable of constructing self-reproducing machines, *Syst. Comput. Japan* **18**:33–40 (1986).
5. K. Morita and S. Ueno, Computation universal models of 2D 16-state reversible partitioned cellular automata, *IEICE Trans. Inf. Syst.* **E75-D**:141–147 (1992).
6. C. G. Langton, Self-reproduction in cellular automata, *Physica D* **10**:135–144 (1984).
7. N. Margolus, Physics-like models of computation, *Physica D* **10**:81–95 (1984).
8. N. Margolus, Crystalline computation, in *Feynman and Computation* (Addison–Wesley, 1998).
9. M. Biafore, Cellular automata for nanometer-scale computation, *Physica D* **70**:415–433 (1994).
10. M. A. Smith, Y. Bar-Yam, Y. Rabin, N. Margolus, T. Toffoli, and C. H. Bennett, Cellular automaton simulation of polymers, *Mat. Res. Soc. Symp. Proc.* **248**:483–488 (1992).
11. D. H. Rothman and J. M. Keller, Immiscible cellular-automaton fluids, *J. Statist. Phys.* **52**:1119–1127 (1988).
12. B. A. Huberman and N. S. Glance, Evolutionary games and computer simulations, *Proc. Natl. Acad. Sci. USA* **90**:7715–7718 (1993).
13. H. Bersini and V. Detours, Asynchrony induces stability in cellular automata based models, in *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, R. A. Brooks and P. Maes, eds. (MIT Press, Cambridge, 1994), pp. 382–387.
14. E. A. Di Paolo, Rhythmic and non-rhythmic attractors in asynchronous random Boolean networks, *BioSystems* **59**:185–195 (2001).
15. M. A. Nowak and R. M. May, Evolutionary games and spatial chaos, *Nature* **359**:826–829 (1992).
16. T. E. Ingerson and R. L. Buvel, Structures in asynchronous cellular automata, *Physica D* **10**:59–68 (1984).
17. H. J. Blok and B. Bergersen, Synchronous versus asynchronous updating in the "game of life," *Phys. Rev. E* **59**:3876–3879 (1999).
18. B. Schönfisch and A. de Roos, Synchronous and asynchronous updating in cellular automata, *BioSystems* **51**:123–143 (1999).
19. S. Kauffman, Metabolic stability and epigenesis in randomly constructed genetic nets, *J. Theor. Biol.* **22**:437–467 (1969).
20. I. Harvey and T. Bossomaier, Time out of joint: Attractors in asynchronous random boolean networks, in *Proceedings of the Fourth European Conference on Artificial Life*, P. Husbands and I. Harvey, eds. (MIT Press, Cambridge, 1997), pp. 67–75.
21. S. F. Edwards and P. W. Anderson, Theory of spin glasses, *J. Phys. F* **5**:965–974 (1975).
22. E. T. Gawlinski, M. Grant, J. D. Gunton, and K. Kaski, Growth of unstable domains in the two-dimensional Ising model, *Phys. Rev. B* **31**:281–286 (1985).
23. H. A. Ceccatto, Effective discrete-time dynamics in Monte Carlo simulations, *Phys. Rev. B* **33**:4734–4738 (1986).
24. K. Kaneko, Period-doubling of kink-antikink patterns, quasiperiodicity in antiferro-like structures and spatial intermittency in coupled map lattice—towards a prelude of a "Field Theory of Chaos," *Prog. Theor. Phys.* **72**:480–486 (1984).
25. H. Chaté and P. Manneville, Collective behaviors in spatially extended systems with local interactions and synchronous updating, *Progr. Theor. Phys.* **87**:1–60 (1992).
26. E. D. Lumer and G. Nicolis, Synchronous versus asynchronous dynamics in spatial distributed systems, *Physica D* **71**:440–452 (1994).
27. P. Marcq, H. Chaté, and P. Manneville, Universal critical behavior in two-dimensional coupled map lattices, *Phys. Rev. Lett.* **77**:4003–4006 (1996).

28. P. Marcq, H. Chaté, and P. Manneville, Universality in Ising-like phase transitions of lattices of coupled chaos, *Phys. Rev. E* **55**:2606–2627 (1997).
29. G. Abramson and D. H. Zanette, Globally coupled maps with asynchronous updating, *Phys. Rev. E* **58**:4454–4460 (1998).
30. J. Rolf, T. Bohr, and M. H. Jensen, Directed percolation universality in asynchronous evolution of spatiotemporal intermittency, *Phys. Rev. E* **57**:R2503–R2506 (1998).
31. N. Gupte, T. M. Janaki, and S. Sinha, Effect of asynchronicity on the universal behavior of coupled map lattices, arXiv:nlin.CD/0205020.
32. S. Goss and J. Deneubourg, Autocatalysis as a source of synchronized rhythmical activity in social insects, *Insects Soc.* **35**:310–315 (1988).
33. D. Cornforth, D. G. Green, D. Newth, and M. Kirley, Do artificial ants march in step? Ordered asynchronous processes and modularity in biological systems, in *Artificial Life VIII, Proceedings of the Eighth International Workshop on Artificial Life*, R. K. Standish, H. A. Abbass, and M. A. Bedau, eds. (MIT Press, Cambridge, 2002), pp. 28–32.
34. W. Freeman, Tutorial on neurobiology: From single neurons to brain chaos, *Int. J. Bifurcation Chaos* **2**:451–482 (1992).
35. D. Green, Simulated fire spread in discrete fuels, *Ecological Modeling* **20**:21–32 (1983).
36. P. Kourtz and W. O'Regan, A model for a small forest fire to simulate burned and burning areas for use in a detection model, *Forest Science* **17**:163–169 (1971).
37. S. Hauck, Asynchronous design methodologies: An overview, *Proc. IEEE* **83**:69–93 (1995).
38. A. Davis and S. M. Nowick, An introduction to asynchronous circuit design, *Technical Report UUCS-97-013*, Computer Science Department, University of Utah, Downloadable from www.cs.columbia.edu/async/publications.html
39. P. Patra and D. S. Fussel, Efficient building blocks for delay insensitive circuits, *Proc. International Symp. on Advanced Research in Asynchronous Circuits and Systems* (1994), pp. 196–205.
40. C. J. Myers, *Asynchronous Circuit Design* (Wiley, 2001).
41. F. Peper, J. Lee, S. Adachi, and S. Mashiko, Laying out circuits on asynchronous cellular arrays: A step towards feasible nanocomputers? *Nanotechnology* **14**:469–485 (2003).
42. J. Lee, F. Peper, S. Adachi, and K. Morita, Universal delay-insensitive circuits with bidirectional and buffering lines, to be published, 2003.
43. P. Patra, S. Polonsky, and D. S. Fussell, Delay-insensitive logic for RSFQ superconductor technology, *Proc. of the Third Int. Symp. on Adv. Res. in Asynchronous Circuits and Systems* (IEEE CS Press, 1997), pp. 42–53.
44. Y. Kameda, S. V. Polonsky, M. Maezawa, and T. Nanya, Self-timed parallel adders based on DI RSFQ primitives, *IEEE Trans. Appl. Superconductivity* **9**:4040–4045 (1999).
45. C. Joachim, J. K. Gimzewski, and A. Aviram, Electronics using hybrid-molecular and mono-molecular devices, *Nature* **408**:541–548 (2000).
46. A. J. Heinrich, C. P. Lutz, J. A. Gupta, and D. M. Eigler, Molecule cascades, *Science* **298**:1381–1387 (2002).
47. C. S. Lent and P. D. Tougaw, A device architecture for computing with quantum dots, *Proc. IEEE* **85**:541–557 (1997).
48. R. P. Cowburn and M. E. Welland, Room temperature magnetic Quantum Cellular Automata, *Science* **287**:1466–1468 (2000).
49. P. Gács, Deterministic computations whose history is independent of the order of asynchronous updating, *Tech. Report*, Computer Science Department, Boston University (1997).
50. P. Orponen, Computing with truly asynchronous threshold logic networks, *Theor. Comp. Sci.* **174**:123–136 (1997).

51. K. Nakamura, Asynchronous cellular automata and their computational ability, *Systems, Computers, Controls* **5**:58–66 (1974)
52. C. L. Nehaniv, Self-reproduction in asynchronous cellular automata, *Proc. NASA/DoD Conf. on Evolvable Hardware, EH'02* (2002), pp. 201–209.
53. T. Toffoli, Integration of the phase-difference relations in asynchronous sequential networks, in *Proc. of the Fifth Colloquium on Automata, Languages, and Programming (ICALP)*, Lecture Notes in Computer Science (LNCS 62), G. Ausiello and C. Böhm, eds. (Springer, 1978), pp. 457–463.
54. J. Lee, S. Adachi, F. Peper, and K. Morita, Asynchronous game of life, to be published, 2003.
55. F. Peper, T. Isokawa, N. Kouda, and N. Matsui, Self-timed cellular automata and their computational ability, *Future Generation Computer Systems* **18**:893–904 (2002).
56. R. M. Keller, Towards a theory of universal speed-independent modules, *IEEE Trans. Comput.* **C-23**:21–33 (1974).
57. J. Lee, S. Adachi, F. Peper, and K. Morita, Embedding universal delay-insensitive circuits in asynchronous cellular spaces, *Fundamenta Informaticae*, accepted, 2003.
58. S. Wolfram, Statistical mechanics of cellular automata, *Rev. Mod. Phys.* **55**:601–644 (1983).
59. E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for Your Mathematical Plays* (Academic Press, 1982).
60. M. Creutz, Confinement and the critical dimensionality of space-time, *Phys. Rev. Lett.* **43**:553–556 (1979).
61. S. Geman and D. Geman, Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images, *IEEE Trans. Pattern Analysis and Machine Intelligence* **6**:721–741 (1984).
62. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equations of state calculations by fast computing machines, *J. Chem. Phys.* **21**:1087–1092 (1953).
63. S. Adachi, J. Lee, and F. Peper, On signals in asynchronous cellular spaces, to be published, 2003.
64. J. A. Brzozowski and J. C. Ebergen, On the delay-sensitivity of gate networks, *IEEE Trans. Comput.* **41**:1349–1360 (1992).
65. A. J. Martin, The limitations to delay-insensitivity in asynchronous circuits, *Proc. 6th MIT Conf. on Advanced Research in VLSI* (MIT Press, Cambridge, 1990), pp. 263–278.
66. P. Reimann, Brownian motors: Noisy transport far from equilibrium, *Phys. Rep.* **361**:57–265 (2002).
67. K. Kitamura, M. Tokunaga, A. H. Iwane, and T. Yanagida, A single myosin head moves along an actin filament with regular steps of 5.3 nanometers, *Nature* **397**:129–134 (1999).
68. Swimming against the tide, news feature, *Nature* **408**:764–766 (2000).