

On a Universal Brownian Cellular Automata with 3 States and 2 Rules

Teijiro Isokawa¹(✉), Ferdinand Peper², Koji Ono¹, and Nobuyuki Matsui¹

¹ Graduate School of Engineering, University of Hyogo, Hyogo, Japan
isokawa@eng.u-hyogo.ac.jp

² Center for Information and Neural Networks,
National Institute of Information and Communications Technology,
Osaka University, Osaka, Japan

Abstract. This paper presents a 3-state asynchronous CA that requires merely two transition rules to achieve computational universality. This universality is achieved by embedding Priese's delay-insensitive circuit elements, called the E-element and the K-element, on the cell space of a so-called Brownian CA, which is an asynchronous CA containing local configurations that conduct a random walk in the circuit topology.

1 Introduction

Cellular Automata (CA) attract increasing attention as architectures for computers with nanometer-scale devices (nano-computers), because their regular structures and local connectivity offer much potential for manufacturing based on molecular self-assembly [1–4]. An obstacle to the realization of nanocomputers is the effect of noise and fluctuations in operating nanometer-scale devices, in which amplitudes of circuit signals could be comparable to those of noise and fluctuations. Assuring the normal operation of circuits will be difficult in the framework of traditional techniques, such as the suppression of noise or the correction of errors caused by noise.

For this reason, alternative approaches to circuit operations need consideration. One possible approach is to make use of noise as information carrier or—more indirectly—for the operations of circuits [5, 6], in a way that is sometimes found in biological systems [7]. When realized in terms of CA, noise-driven computation is described by the term *Brownian Cellular Automata (BCA)* [8]. BCA are a type of asynchronous CA, where certain local configurations propagate randomly in the cellular space, resembling Brownian motion. The BCA in [8] is proven computational universal, by embedding so-called Brownian circuits [9] on the cell space. The number of states in a cell is 3 in this model and the number of transition rules is 3, which is much less than comparable models in asynchronous CA with computational universality. The resulting decrease in the complexity of a cell is very useful for the efficient implementation of nanometer-scale devices.

K. Ono has been with Glory Limited, Japan, since 2011.

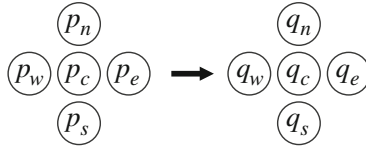


Fig. 1. Transition on BCA

This paper presents a computational universal BCA, in which the number of cells' state is 3, like in [8], but in which the number of transition rules is decreased to 2. Computational universality in the proposed BCA is shown by embedding a set of previously known delay-insensitive circuit elements, called K-element and E-element [10], on the cell space.

2 Preliminaries

2.1 Brownian Cellular Automaton

A BCA [8] is a two-dimensional asynchronous CA of identical cells, each of which can assume one of a finite number of states at a time. Cells undergo transitions in accordance with transition rules that operate on each cell and its direct four neighbors, shown in Fig. 1. The rules are of a type called Von Neumann neighborhood aggregate rules. In a BCA, transitions of the cells occur at random times, independent of each other. Furthermore, it is assumed that neighboring cells of the cells being in transition never undergo transitions at the same time to prevent a situation in which such cells simultaneously write different states into the same location.

We assume that the transition rules are rotational symmetric, i.e., one transition rule has four rotated analogues. Consequently, when we represent the transition in Fig. 1 as

$$(p_c, p_n, p_e, p_s, p_w) \rightarrow (q_c, q_n, q_e, q_s, q_w), \quad (1)$$

the following three rules also exist:

$$\begin{aligned} (p_c, p_e, p_s, p_w, p_n) &\rightarrow (q_c, q_e, q_s, q_w, q_n) \\ (p_c, p_s, p_w, p_n, p_e) &\rightarrow (q_c, q_s, q_w, q_n, q_e) \\ (p_c, p_w, p_n, p_e, p_s) &\rightarrow (q_c, q_w, q_n, q_e, q_s) \end{aligned}$$

2.2 Computational Elements

A few decades ago Priese [10] proposed circuit elements from which arbitrary delay-insensitive circuits can be constructed. Called the E-element and

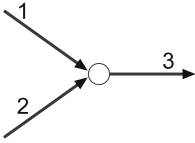


Fig. 2. K-element

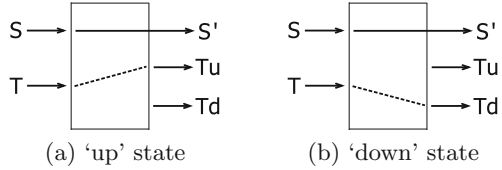


Fig. 3. E-element in (a) ‘up’ state and (b) ‘down’ state

K-element [10], these elements—schematically shown in Figs. 2 and 3—are universal, forming a base for the construction of a sequential automaton. The circuits constructed from E-elements and K-elements have in common that they employ only one signal at a time. Though inefficient, this is sufficient to guarantee universality.

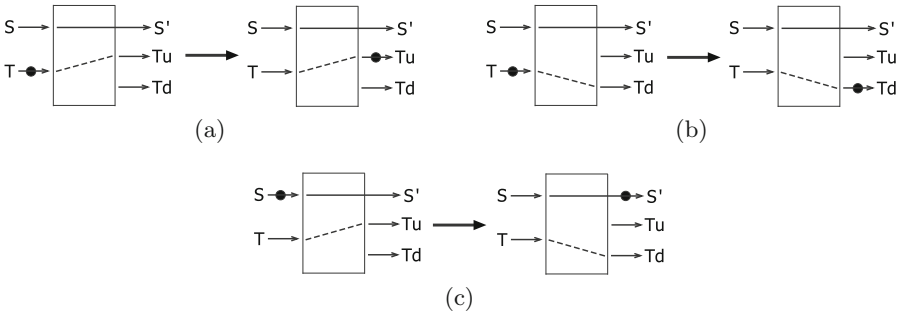


Fig. 4. Operations of an E-element: (a) when in the ‘up’ state, (b) when in the ‘down’ state, and (c) changing state upon receiving an input signal on wire S. A token (blob) on a wire denotes a signal.

The K-element has two input wires and one output wire and it accepts a signal coming from either input wire and outputs it to the output wire.

The E-element is an element with two input wires (S and T) and three output wires (S' , T_u , and T_d), as well as two internal states (‘up’ or ‘down’). Input from wire T will be redirected to either of the output wires T_u or T_d , depending on the internal state of the element: when this state is ‘up’ (resp. ‘down’), a signal on the input wire T flows to the output wire T_u (resp. T_d) as in Fig. 4(a) (resp. Fig. 4(b)). By accepting a signal from input wire S, an E-element changes its internal state from ‘up’ to ‘down’ or from ‘down’ to ‘up’, after which it outputs a signal to output wire S' , as shown in Fig. 4(c).

3 Implementing Computational Elements on Brownian Cellular Automaton

3.1 Basic Elements

A cell in the proposed BCA can be in one of the 3 states, as shown in Fig. 5. Figure 6 shows two transition rules used in the proposed CA. To construct the delay-insensitive circuit elements, we first define the basic elements on the above-mentioned BCA, called *signal*, *signal line*, *terminator*, *hub*, *crossover*, and *switch*.

Symbol	"Blank"	○	●
State	0	1	2



Fig. 5. The symbols used to represent the states of a cell

Fig. 6. Transition rules

The most basic elements are the *signal* and the *signal line*. A signal can travel bidirectionally along a signal line (Fig. 7(a)). This is implemented on the BCA by appropriate placements of state 1, as shown in Fig. 7(b). The transition rule #1 is used for moving a signal to and fro on a signal line, whereby the direction of the signal is determined by which cell first undergoes a transition. In the example in Fig. 7(b), both ends of a signal line are terminated by *terminators*, thus a signal travels between terminators.

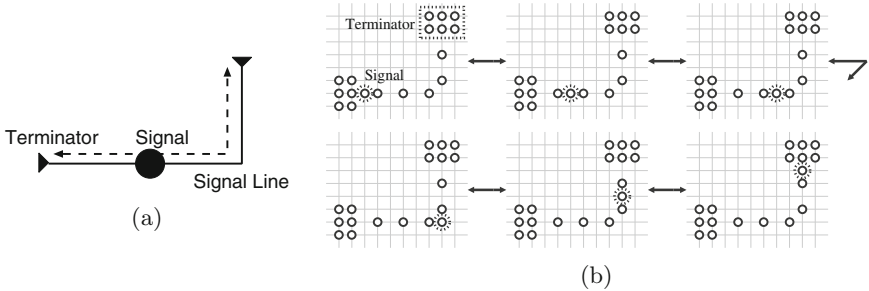


Fig. 7. (a) Signal, Signal line, and Terminators, (b) Their implementations on BCA, where a cell with a dotted circle will undergo the transition.

A *hub* is used for branching a signal line (Fig. 8(a)). The signal on the port *a* goes out from *a*, *b* or *c*. A configuration of a hub on BCA is shown in Fig. 8(b). The transition rule #1 is also used for this element.

A *crossover* element (Fig. 9(a)) accepts a signal at port *N* (or *S*) and produces an output signal from port *S* (or *N*, resp.). In a similar way, the ports *W* and *E* are connected to each other. Figure 9(b) shows a configuration of a crossover on BCA.

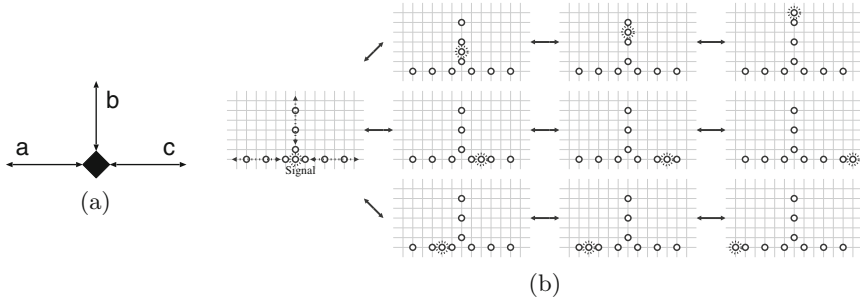


Fig. 8. (a) a Hub and (b) its implementation on BCA

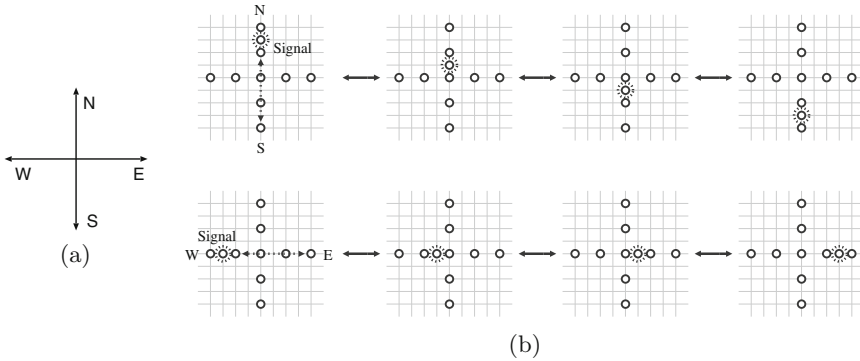


Fig. 9. (a) a Crossover and (b) its implementation on BCA

The next circuit element requires some kind of switching behavior, for which the basic mechanism works like in Fig. 10. Using transition rule #1, this mechanism drags the right end of a signal line on which no terminator exists in the right direction. Dragging of the right end continues until the signal arrives at the other signal line, somewhere to the right, resulting in the line end to become attached to the left end of this line. This effect of dragging the end of a line towards another line can also be used in combination with curves of a line, in which a signal turns to the left or to the right. It is used in an element called *switch* (Fig. 11). A switch element has two ports, both of which can be used for input or output, and it takes one of two states, the *R*-state and the *L*-state. An arrow in a switch represents the direction of a signal that can be passed: in the *R*-state (Fig. 11(a)), a signal at port *W* can go out from port *E* but a signal at port *E* never passes through the element. Similarly, a signal at port *E* can go out from port *W* in the *L*-state (Fig. 11(b)). On acceptance of a signal, the switch element changes its state from *L* to *R* (or *R* to *L*, resp.) (see Fig. 11(c)). A configuration of a switch and its switching behavior is shown in Fig. 12.

There is another switching element, called a *line selector* and shown in Fig. 13. This element has six input/output ports and takes one of two states, the *U*-state

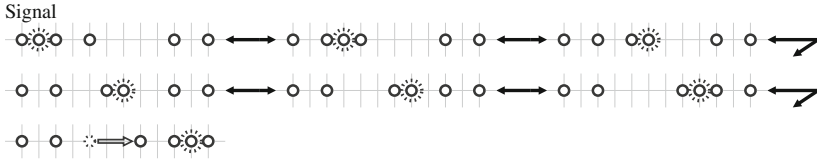


Fig. 10. Dragging state 1 from one signal line to a signal line right of it.

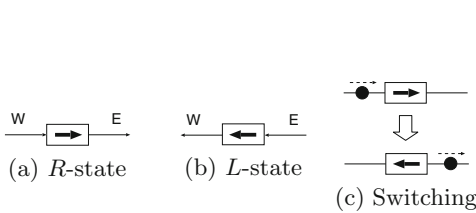


Fig. 11. A Switch element (a) *R-state*, (b) *L-state*, and (c) switching from *L* to *R*.

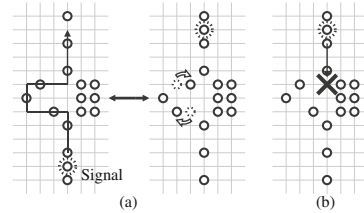


Fig. 12. A BCA implementation for switch element

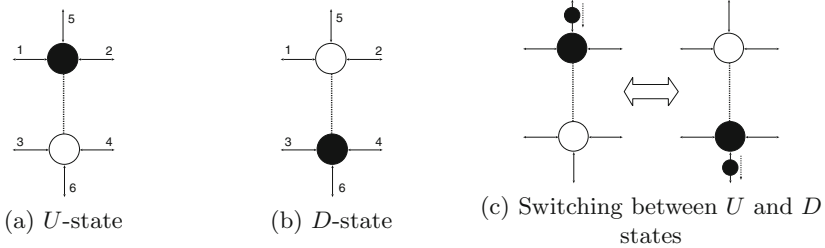


Fig. 13. A line selector element (a) *U-state*, (b) *D-state*, and (c) switching by a signal.

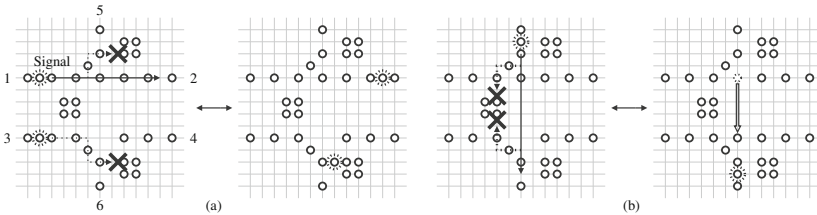


Fig. 14. A line selector element implemented on BCA

and the *D-state*. A line selector in the *U-state* (Fig. 13(a)) connects the port 1 and the port 2 by a signal line, thus allowing a signal on port 1 (or 2) to move to port 2 (or 1). In this state, ports 3 and 4 are not connected, i.e., a signal on port 3 never goes out from 4 and vice versa. When this element is in the

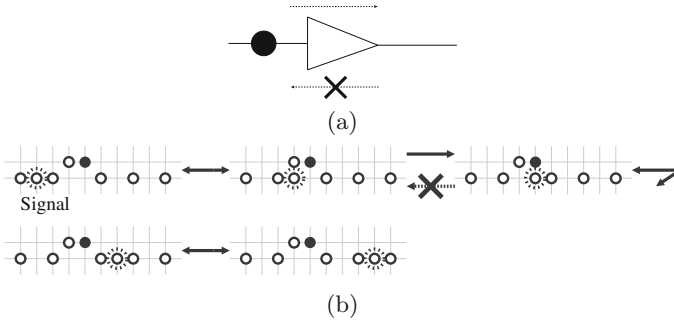


Fig. 15. (a) A ratchet and (b) its implementation on BCA

D-state (Fig. 13(b)), on the other hand, it connects port 3 and port 4, but it does not connect port 1 and port 2. The switching of the state of a line selector is conducted by a signal traveling from port 5 to port 6. When a line selector is in state *U*, it accepts a signal from port 5, then produces a signal at port 6, and finally changes its state to *D* (Fig. 13(c)). Similarly, a line selector in state *D* changes its state to *U* when it accepts a signal at port 6.

The behavior of a line selector can be realized on the proposed BCA by utilizing the dragging behavior in Fig. 10 on an appropriate configuration of state 1s. Figure 14 shows a line selector implemented on BCA.

The last of the basic elements is called a *ratchet*. Figure 15 shows a ratchet and its configuration on BCA. This element has one input port and one output port. A signal on the input port will go out from the output port, but this signal cannot return to the input port from the output port. For realizing this mechanism, an additional state (i.e. state 2) and an additional transition rule (rule #2) are used (see Fig. 15(b)).

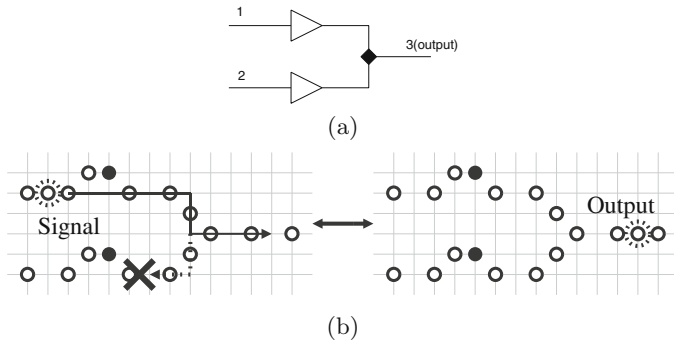


Fig. 16. (a) A K-element by basic elements and (b) its implementation on BCA

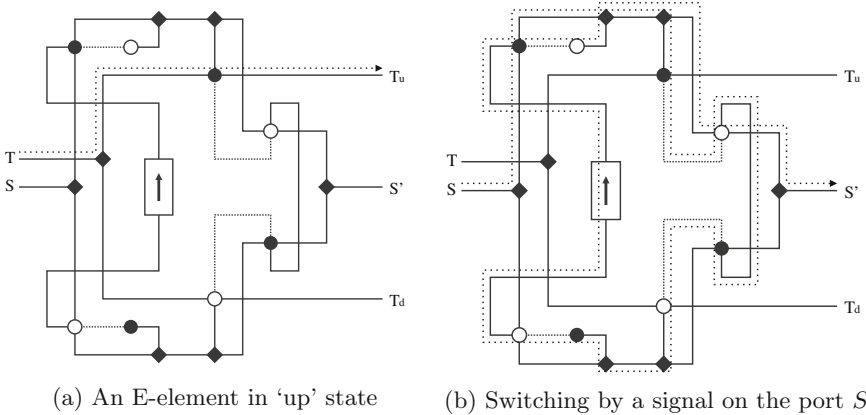


Fig. 17. An E-element constructed by basic elements, where it is in the state 'up'. (a) Trajectory of a signal on the port T . This results in the signal going out from T_u . (b) Trajectory of a signal on the port S . The state of the element changes to 'down' and the signal goes out from the port S' .

3.2 Constructing Computational Elements

To ensure the computational universality on the proposed BCA, we implement two elements—the E-element and the K-element—as configurations on BCA. Implementing a K-element is straightforward. Figure 16(a) shows a construction of a K-element in terms of the basic elements. It uses one hub for combining two

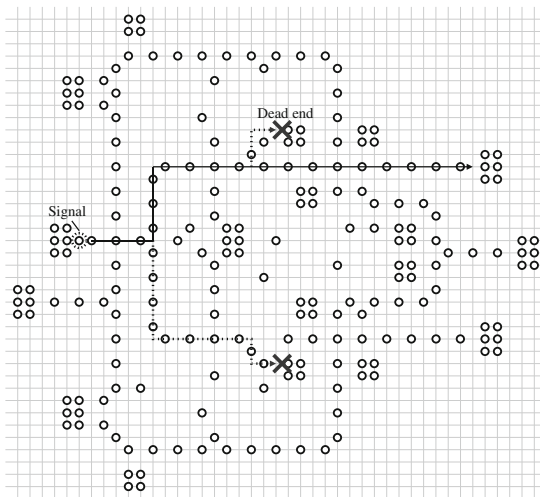


Fig. 18. An implementation of an E-element on BCA. A solid line with an arrow represents a trajectory of a signal.

input lines, and two ratchets for preventing the signal going out from the input ports. A signal on either 1 or 2 will be output to port 3. A configuration of a K-element on the BCA is shown in Fig. 16(b).

A construction of the E-element is shown in Fig. 17(a), where it is in the ‘up’ state. The states of four line selectors and a switch in the element determine the state of this E-element: in the case of the ‘up’ state, a signal at the port T will eventually move to port T_u . A trajectory of a signal on port T is shown as a dashed line in Fig. 17(a). Switching the state of an E-element is accomplished by a signal on port S , resulting in this signal going out from port S' . Figure 17(b) shows a trajectory of a signal that is input to port S and goes out from S' . All the states of the switch and the line selectors are changed by this signal.

A BCA implementation of an E-element is shown in Fig. 18, where the state of this element is ‘up’. A solid line with an arrow in this figure shows a trajectory of a signal on the port T , which corresponds to a dashed line in Fig. 17(a). Transitions of switching from the state ‘up’ to ‘down’ are shown in Fig. 19.

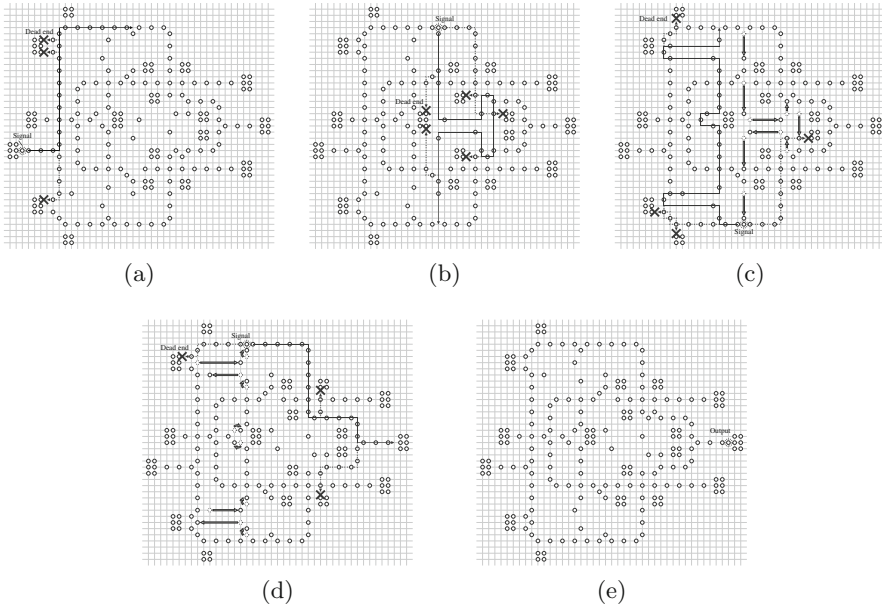


Fig. 19. Switching the state of an E-element

By arranging E-elements and K-elements and connections between them in appropriate ways on the proposed BCA, we can realize any circuit consisting of these elements. Thus, the proposed BCA is computationally universal.

4 Conclusion

This paper presents a 3-state computationally universal Brownian CA, in which the number of transition rules is two. This is one rule less than in the Brownian CA model in [8].

There is some limited room for further reduction of the number of states and the number of rules. Obviously, the limit is two states and one rule. Interestingly, the E-element, which is the most complicated of the two circuit elements, requires exactly this number of states and rules for its implementation. It is for two reasons that a third state and a second rule are necessary. The first reason is that a ratchet appears to require such an addition of one state and one rule. Rule #1 is symmetric in itself, and it is likely unable to provide for the asymmetry of the ratchet. Regarding the need for an additional state, it is unclear at this stage whether a ratchet can be implemented without relying on it, so further research is needed to provide more clarity. The second reason why two states and one rule appear insufficient is that a 1-cell shift is caused by a turn left or right of a line, and this shift makes it impossible to line up certain input and output lines. The construction of the ratchet provides for this shift in a signal line, but, as stated above, an additional state and rule are necessary for this.

Acknowledgment. This work was supported by a Grant-in-Aid for Scientific Research on Innovative Areas “Molecular Robotics” (No. 15H00825) of The Ministry of Education, Culture, Sports, Science, and Technology, Japan.

References

1. Biafore, M.: Cellular automata for nanometer-scale computation. *Physica D* **70**, 415–433 (1994)
2. Durbeck, L.J.K., Macias, N.J.: The cell matrix: an architecture for nanocomputing. *Nanotechnology* **12**, 217–230 (2001)
3. Peper, F., Lee, J., Adachi, S., Mashiko, S.: Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers? *Nanotechnology* **14**, 469–485 (2003)
4. Peper, F., Lee, J., Isokawa, T.: Cellular nanocomputers: a focused review. *Int. J. Nanotechnol. Mol. Comput.* **1**, 33–49 (2009)
5. Kish, L.B.: Thermal noise driven computing. *Appl. Phys. Lett.* **89**(14), 144104 (2006)
6. Dasmahapatra, S., Werner, J., Zauner, K.P.: Noise as a computational resource. *Int. J. Unconventional Comput.* **2**(4), 305–319 (2006)
7. Yanagida, T., Ueda, M., Murata, T., Esaki, S., Ishii, Y.: Brownian motion, fluctuation and life. *Biosystems* **88**(3), 228–242 (2007)
8. Lee, J., Peper, F.: Efficient computation in brownian cellular automata. In: Peper, F., Umeo, H., Matsui, N., Isokawa, T. (eds.) *Proceedings of 4th International Workshop on Natural Computing*, vol. 2, pp. 47–56. Springer, Tokyo (2009)
9. Peper, F., Lee, J., Carmona, J., Cortadella, J., Morita, K.: Brownian circuits: fundamentals. *ACM J. Emerg. Technol. Comput. Syst.* **9**(1), 1–24 (2013). Article no. 3
10. Priese, L.: Automata and concurrency. *Theoret. Comput. Sci.* **25**(3), 221–265 (1983)