Computing by Swarm Networks

Teijiro Isokawa[1], Ferdinand Peper[1,2], Masahiko Mitsui[1],
Jian-Qin Liu[3], Kenichi Morita[4], Hiroshi Umeo[5],
Naotake Kamiura[1], and Nobuyuki Matsui[1]

[1] Division of Computer Engineering, University of Hyogo, Japan
isokawa@eng.u-hyogo.ac.jp
[2] Nano ICT Group,
National Institute of Information and Communications Technology, Japan
[3] Biological ICT Group,
National Institute of Information and Communications Technology, Japan
[4] Dept. of Information Engineering, Hiroshima University, Japan
[5] Dept. of Computer Science, Osaka Electro-Communication University, Japan

Abstract. Though the regular and fixed structure of cellular automata greatly contributes to their simplicity, it imposes a strict limitation on the applications that can be modeled by them. This paper proposes *swarm networks*, a model in which cells, unlike in cellular automata, have irregular neighborhoods. Timed asynchronously, a cell in this model acts like an *agent* that can dynamically interact with a varying set of other cells under the control of transition rules. The configurations in which cells are organized according to their neighborhoods can move around in space, following simple mechanical laws. We prove computational universality of this model by simulating a circuit consisting of asynchronously timed circuit modules. The proposed model may find applications in nanorobotic systems and artifical biological systems.

1 Introduction

Nanorobots, Artificial Cells, Smart Dust—all these models have in common a large number of distributed units that interact with each other to conduct certain tasks. Like in Cellular Automata (CA), the underlying units are relatively simple—usually being nothing more than Finite State Machines—but unlike in CA the units are more dynamic in the way they interact. They form swarms of agents that communicate with each other through dynamic networks of interconnections. How can we characterize the functionality of such swarms? Will their less-rigid communication structures cause a loss of functionality relative to CA models with comparable complexity? Or will swarm networks be more powerful through their more flexible way of interaction?

A useful measure of power—useful at least in the world of computer scientists– is whether a model is computationally universal. This measure basically separates "interesting" models from the "uninteresting" ones, forming the major motivation in the last century to characterize models in terms of computability. Universal Turing Machines [1] are well-known in this context, but other models have been proposed too [2]. In the context of CA, computational universality is often proved by embedding logic circuits on cellular space. This requires a CA to simulate a universal set of primitives, like the AND-gate and the NOT-gate, as well as to simulate signals being propagated between these primitives. Some well-known computationally universal CA are found in [3,4,5,6] when the timing model is synchronous—implying the simultaneous update of all cells at each step. In an asynchronous model, on the other hand, update of cells is randomly timed. Universality of asynchronous CA is proven in a similar way as with synchronous CA, i.e., by laying out circuits on cellular space, be it that different sets of primitives are used to compensate for the lack of clock signals [7,8,9,10,11].

Swarms have been researched in contexts varying from insects [12] to swarm robotics [13], and from distributed sensor networks [14] to groupware systems [15]. Stevens [16] has proposed a swarm-based system that is able to replicate itself. Simulated on a computer, the agents in this model are divided in different types, each with a different functionality. There are agents that conduct boolean operations, but also agents that exert forces in certain directions, and so on, and the agents move in 2-dimensional continuous space according to Newton-like laws. Each agent has four terminals through which it can be connected to and exchange integer values with other agents. Through these connections, agents can be organized in certain configurations, that act like a kind of “organisms”, which have more complicated functionalities than individual agents.

In this paper we present a swarm network model in which all agents are identical, like the cells in CA but unlike in Stevens’ model. The functionalities of the agents are determined by their states as well as by the patterns by which they are mutually interconnected. Based on these agents, we construct two circuit elements that form a universal set of primitives in the class of delay-insensitive circuits [17], i.e. circuits robust to delays in their wires and primitives. This result implies that any arbitrary delay-insensitive circuit can be constructed from the agents, proving the computational universality of the model.

This research promises applications in which simplicity of agents is important, while the cooperative actions of the agents are sufficiently powerful to result in interesting behavior. Nanorobotics is one particular application that comes to mind: the tiny robots in such an application face severe restrictions in their complexity; yet, combined in swarms of nanorobots, they should have a certain minimal functionality to be of use. Sensor networks may be another application, in which sensor agents derive added functionality from the mutual cooperation in their sensing behavior, for example to measure gradients in certain physical observables.

2 Computational Elements

A few decades ago Priese [17] proposed circuit elements from which arbitrary delay-insensitive circuits can be constructed. Called the E-element and K-element [17], these elements—schematically shown in Figs. 1 and 2—are universal, forming a base for the construction of a sequential automaton. The circuits constructed from

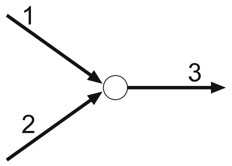

Fig. 1. K-element

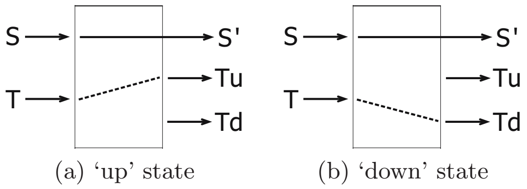

Fig. 2. E-element in (a) 'up' state and (b) 'down' state

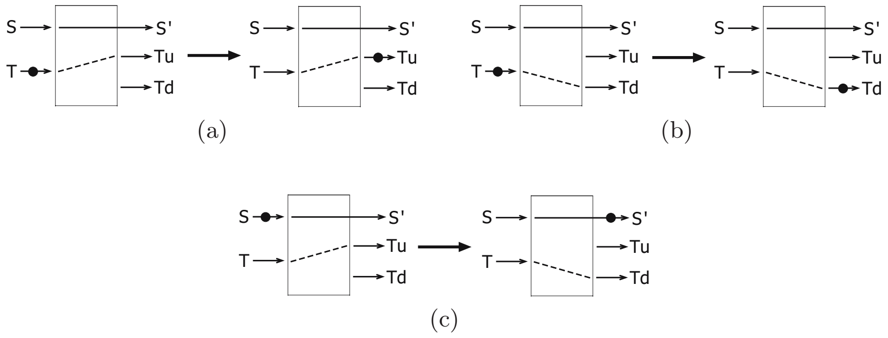

Fig. 3. Operations of an E-element: (a) when in the 'up' state, (b) when in the 'down' state, and (c) changing state upon receiving an input signal on wire S. A token (blob) on a wire denotes a signal.

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.

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. 3(a) (resp. Fig. 3(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. 3(c).

3 Model of the Agents

Contained in two-dimensional space, agents in the proposed swarm networks model have a circular shape, the outside of which has six terminals attached at identical distances from each other (Fig. 4). Agents are connected to each other via these terminals, which are used to exchange input and output between agents. The terminal colored black in Fig. 4 indicates that it forms a connection with another agent. Each agent is assumed to be a Mealy-type finite automaton, with an internal state denoted by a symbol in it (Fig. 4). The functionality of an agent is determined—apart from the agent's state—by the connection pattern of the agent to other agents. So, an agent being connected to two agents at opposite terminals, for example, has a different functionality than an agent connected to three agents via adjacent terminals.

An agent's functionality is mostly expressed in terms of logical transitions, but it may also contain a physical component. For certain patterns at which an agent is connected to other agents, the agent may experience a force exerted from a certain terminal, pulling it in a certain direction. The space containing the agents satisfies simple mechanical laws. Apart from the abovementioned forces, there are forces between terminals interconnected to each other. Modeled as springs, the connections between terminals exert a repulsive force between terminals very near to each other, and an attractive force between terminals more remote from each other. So, connections are elastic. Communication is not only limited to terminals connected to each other, but may also take place between two terminals that are unconnected but very near to each other, to the extent that the two terminals are at a distance that is less than the distance between two adjacent terminals within an agent. The states and output of the agent are determined by a transition function. This function has as its domain the agent's internal state, the input values from the I/O terminals, and the connection pattern of the I/O terminals. The output domain of the transition function covers the agent's internal state, the output values to the I/O terminals, and whether a force is exerted to the agent. Formally, the transition function f is defined as:

$$f(q, \boldsymbol{i}, \boldsymbol{c}) \rightarrow (q', \boldsymbol{o}, m) \tag{1}$$

where q and q' are the internal states before and after the transition, respectively, $\boldsymbol{i} = \{i_0, \cdots, i_5\}$ is the set of the input values on the I/O terminals, $\boldsymbol{o} = \{o_0, \cdots, o_5\}$ is the set of values output to the I/O terminals. The connection pattern is denoted by $\boldsymbol{c} = \{c_0, \cdots, c_5\}$. The value of m in the output of the transition function determines whether a force is exerted upon the agent. We assume that the transition rules are rotation symmetric, i.e., one transition rule exists in six varieties, which are rotated analogues of each other. An illustrative transition rule is depicted in Fig. 5, where the direction of the exerted force is indicated by the dotted arrow.

4 Building Circuits by Swarm Networks

To establish the computational universality of the model, we show how the K- and E-elements can be constructed by groups of agents. An agent in the model has one of two states (q_1 or q_2) and each of its I/O terminals inputs and outputs a number from the set $\{1, 2\}$. There is also another type of agent, called *wall agent*, that is passive. Wall agents are lined up into structures that form boundaries between which configurations of the normal agents may move around. Represented as black circles in the figures, wall agents constitute the isolating walls

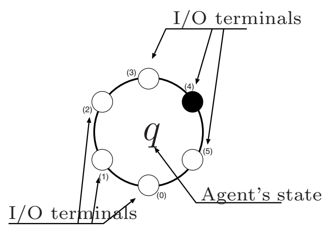

Fig. 4. Individual Agent. The terminals are labeled by the numbers between brackets.

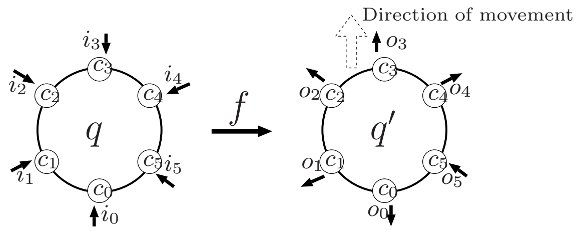

Fig. 5. Transition rule for an agent

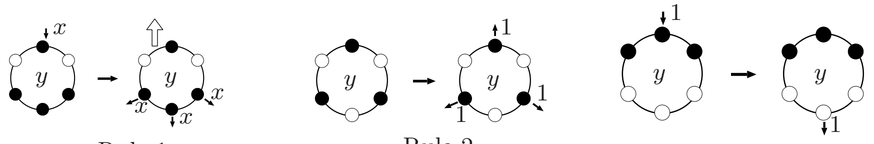

Fig. 6. Transition rules for a signal: rule 1 facilitates the movement of a signal, rule 2 provides a constant source of 1-values driving the processes in the signal, and rule 3 provides 1-outputs at the tail of the signal to be sensed by the switching bar in the E-element.

of the wires in the circuits to be constructed. Signals travel in the circuit along these wires.

The behavior of agents is governed by transition rules, which are applied to the agents according to an asynchronous updating mode (random timing). Fig. 6 shows three transition rules that are used in the operation of a signal, the configuration of which is shown in Fig. 7. The symbol x in rule (1) denotes an input and output value being either 1 or 2. When, for example, the input value at terminal 3 of an agent is 1, the same value 1 is output at terminals 0, 1, and 5. This rule also facilitates the exertion of a force such that the agent moves to the north. There is one agent in a signal that is governed by rule (1): it is at the inner part of the signal, and denoted by the symbol (b) in the configuration constituting the signal (Fig. 7). Transition rule (2) applies to another agent in the inner part of the signal, which is labeled by the symbol (a) in Fig. 7. This agent provides a constant stream of 1-values output to other agents in the signal, such as the above agent labeled by (b). These 1-values being output are also received by agents that behave according to rule (3), and these agents respond with a 1-value output to their opposite terminal. This 1-value being output will in its turn be transmitted to the E-element when the signal passes through it, as a result of which the E-element's state will be flipped, as we will see later.

The turn of a wire resp. crossing of two wires can be implemented in a straightforward way, i.e., by appropriate configurations of wall agents, which guide the

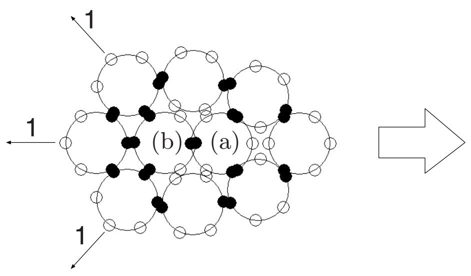

Fig. 7. A configuration of a signal. The big arrow right of the configuration denotes the direction of the signal.

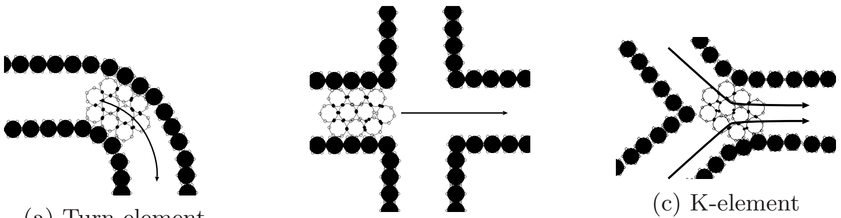

Fig. 8. Configurations of circuit element by wall agents

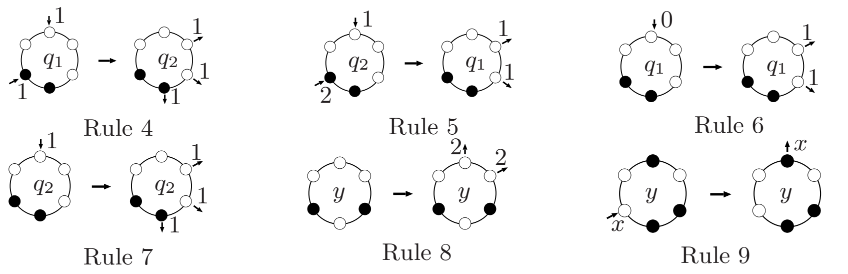

Fig. 9. Transition rules for operating an E-element

signals in accordance with the wall's shape (Figs. 8(a) resp. 8(b)). The temporary absence of wall structures at the time of a signal's crossing does not affect the proper passing of the signal through the crossing, since the force exerted on a signal pulls it across this momentary lapse of the walls. The K-element can be constructed by wall agents in a similar way as the turn and crossing elements (Fig. 8(c)): a signal from either input wire will be guided to the output wire by the wall agents.

For the construction of an E-element six more transition rules are required, which are shown in Fig. 9. The agents to which rules (4), (5), (6), and (7) apply all have the same connection patterns, but each of these rules applies to different patterns of input and output values from other agents. The symbol 0 at terminal

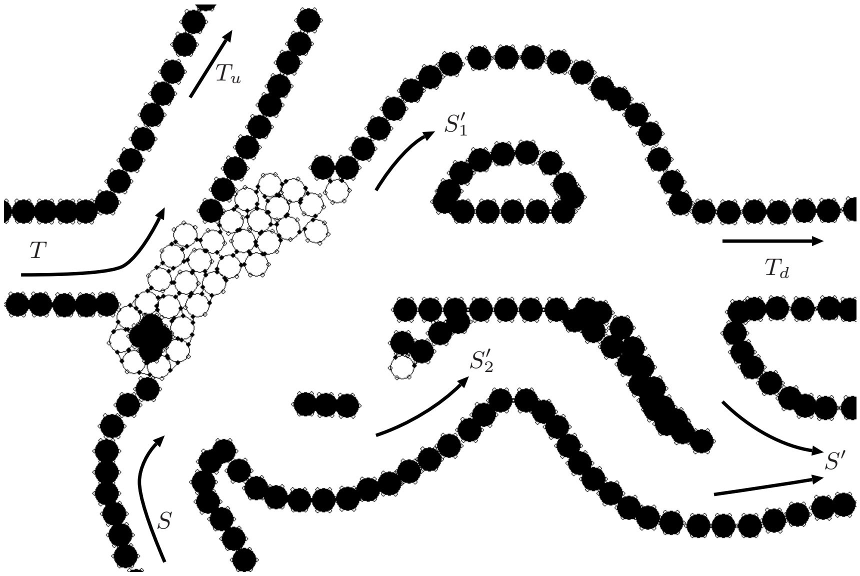

Fig. 10. A configuration of agents representing an E-element

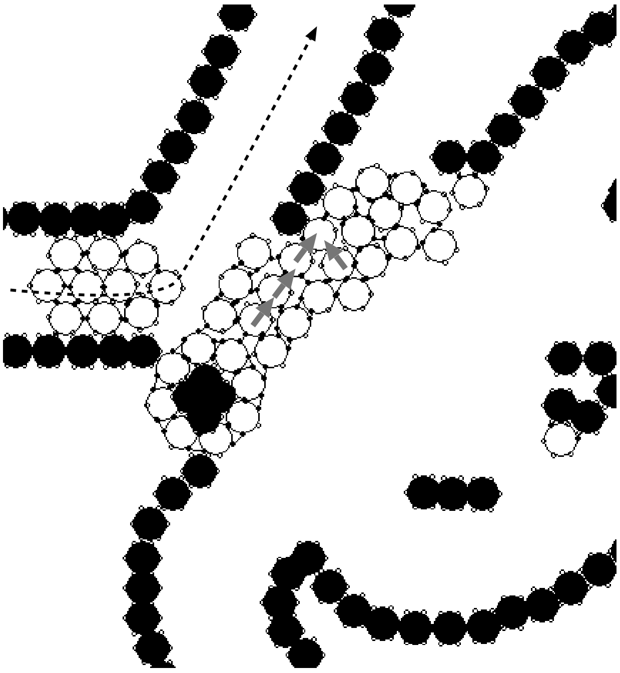

Fig. 11. Flow of a signal in an E-element in the state ‘up’

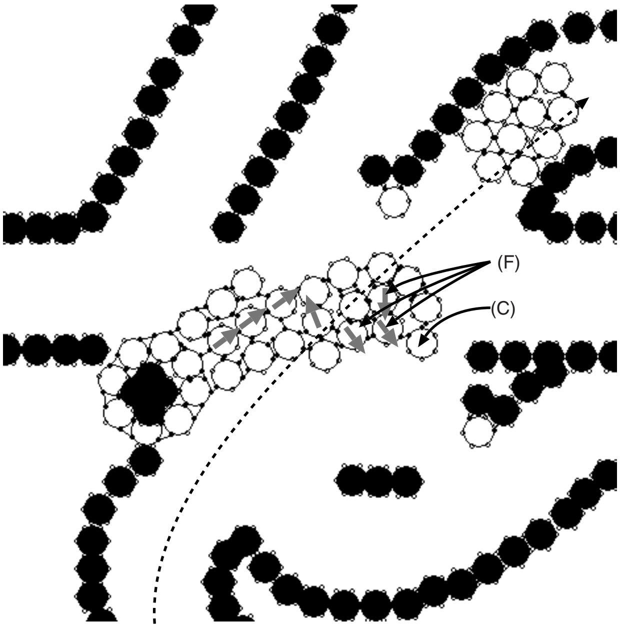

Fig. 12. Switching of an E-element from the state 'up' to the state 'down' as the result of a signal being input from terminal S

3 in rule (6) denotes the condition that no values from another agent are input to this terminal.

Fig. 10 shows a realization of an E-element by agents. Signals in this element flow according to the direction of the arrows. The rotation bar in this configuration, which hinges around a fixed post of wall agents, indicates the state of the E-element by its position. In Fig. 10 the state is 'up'. A signal from input wire T exits from either output wire T_u or T_d, depending on the direction of the rotation bar, without changing the state of the E-element. A signal input to wire S also passes through the E-element and leaves from either output wire S'_1 or S'_2, before eventually being merged into one output wire S'. In this case, the state of the E-element changes as the 1-outputs emanating from a signal's tail result in the rotation bar being flipped.

Fig. 11 shows a signal passing through an E-element in state 'up' in more detail, with the directions of the forces exerted on the agents inside of the rotation bar indicated by gray arrows. Though the signal passing through the E-element touches the rotation bar, and thus exerts yet another force on it, the integrity of the rotation bar (and thus its shape) is maintained as a result of the pulling forces inside the bar. Switching of the E-element takes only place if the signal's tail touches the tip of the rotation bar, but in this case this will not happen, since there is a wall in between to prevent this.

When a signal enters the E-element from input terminal S, like in Fig. 12, its tail will touch the tip of the rotation bar on the signal's exit. This results in the transmission of a 1-value from the signal to agent (C) in the bar's tip in Fig. 12. The chain of reactions caused by this between agents in the rotation bar will then effectuate state changes in some of these agents. Accompanying the state changes is an increase in the number of agents on which downward forces are exerted. To be exact, three agents will experience a downward force, and these agents are indicated by the symbol (F) in Fig. 12. The rotation bar will then rotate downward as a result of these forces. The opposite of this process—switching the E-element's state back to state 'up'—is accomplished in a similar way, by inputing a signal to S, which will then pass S'_2 in Fig. 10, in the process touching the rotation bar's tip. The opposite will then happen: the downward forces will be switched off and the bar will move upwards again.

5 Conclusions and Discussion

This paper presents a model of swarm networks and shows how to conduct universal computation by groups of agents in these networks. An agent is a two-state Mealy-type finite automaton with six input/output terminals, some of which are connected to other agents' terminals. The state of an agent, the output values of its terminals, and the connectivity of each agent determine the functionality of the agent; these variables are thus directly reflected by the transition rules in the model. Agents are similar to cells in cellular automaton models, except that the interconnection structure between agents is irregular. Universal computation is achieved in this model by nine transition rules, through the simulation of the asynchronously timed K and E circuit elements, as well as through the simulation of signal propagation between these elements. Simulations on a computer reveal that the proposed model behaves in a way that somehow resembles biological phenomena. The elastic nature of the connections between the agents appears an important ingredient in this context, as it results in an efficient distribution of the pulling forces among agents connected to each other.

The proposed model may be useful for the realization of computational devices based on biological mechanisms or other physical nanometer-scale interactions. The agents could for example be implemented in terms of proteins. This includes motor proteins, i.e., proteins that facilitate the transport of certain chemical substances inside organisms. It is well-known that proteins can be bound to other proteins, like with our agents, and that such bindings result in new properties and behavior of the formed components [18,19]. A protein can be thought of as being in a certain state through the addition of a phosphorus molecule: when such a molecule is present, we speak of a *phosphorylated* protein, otherwise of a *dephosphorylated* protein. The state of a protein can be influenced by the state of other proteins in its vicinity, according to so-called *domain-specific reactions*, a domain in a protein corresponding to a terminal in an agent. These reactions tend to be strongly dependent on the bindings of the protein to other proteins, in a similar way as the interconnection pattern of an agent with other agents influences the agent's functionality. Though space does not allow us to give specific biological implementations of the agents, the richness of interactions between proteins provide ample inspiration toward the realization of this.

References

1. Turing, A.: On computable numbers, with an application to the entscheidungsproblem. Proc. London Math. Soc. 2(42), 230–265 (1936)
2. Church, A.: An unsolvable problem of elementary number theory. American Journal of Mathematics 58(2), 345–363 (1936)
3. Neumann, J.V.: Theory of Self-Reproducing Automata. University of Illinois Press, Champaign (1966)
4. Banks, E.: Universality in cellular automata. In: IEEE 11th Ann. Symp. on Switching and Automata Theory, pp. 194–215 (1970)
5. Codd, E.F.: Cellular Automata. Academic Press, Inc., Orlando (1968)
6. Serizawa, T.: Three-state Neumann neighbor cellular automata capable of constructing self-reproducing machines. Syst. and Comput. in Japan 18(4), 33–40 (1987)
7. Adachi, S., Peper, F., Lee, J.: Computation by asynchronously updating cellular automata. Journal of Statistical Physics 114(1/2), 261–289 (2004)
8. Lee, J., Adachi, S., Peper, F., Morita, K.: Embedding universal delay-insensitive circuits in asynchronous cellular spaces. Fundamenta Informaticae 58(3/4), 295–320 (2003)
9. Lee, J., Adachi, S., Peper, F., Mashiko, S.: Delay-insensitive computation in asynchronous cellular automata. Journal of Computer and System Sciences 70(2), 201–220 (2005)
10. Peper, F., Lee, J., Adachi, S., Mashiko, S.: Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers? Nanotechnology 14(4), 469–485 (2003)
11. Peper, F., Lee, J., Abo, F., Isokawa, T., Adachi, S., Matsui, N., Mashiko, S.: Fault-tolerance in nanocomputers: A cellular array approach. IEEE Trans. Nanotechnology 3(1), 187–201 (2004)
12. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Santa Fe Institute Studies on the Sciences of Complexity. Oxford University Press, USA (1999)
13. Bayindir, L., Sahin, E.: A review of studies in swarm robotics. Turk J. Elec. Engin. 15(2), 115–147 (2007)
14. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A survey on sensor networks. IEEE Communications Magazine 40(8), 102–114 (2002)
15. ter Beek, M., Ellis, C., Kleijn, J., Rozenberg, G.: Synchronizations in team automata for groupware systems. Comput. Supported Coop. Work 12(1), 21–69 (2003)
16. Stevens, W.: Simulating self-replicating machines. Journal of Intelligent and Robotic Systems 49(2), 135–150 (2007)
17. Priese, L.: Automata and Concurrency. Theoretical Computer Science 25(3), 221–265 (1983)
18. Krauss, G.: Biochemistry of signal transduction and regulation. Wiley-VCH, Weinheim (2008)
19. Liu, J.Q., Shimohara, K.: Biomolecular computation for bionanotechnology. Artech House, Boston (2007)