Information and Computation 274 (2020) 104539

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Turing-completeness of asynchronous non-camouflage cellular automata

Tatsuya Yamashita [a], Teijiro Isokawa [b], Ferdinand Peper [c,b], Ibuki Kawamata [d], Masami Hagiya [a,*]

[a] Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
[b] Graduate School of Engineering, University of Hyogo, Himeji, Japan
[c] Center for Information and Neural Networks, National Institute of Information and Communications Technology, and Osaka University, Osaka, Japan
[d] Department of Bioengineering and Robotics, Graduate School of Engineering, Tohoku University, Sendai, Japan

ARTICLE INFO

Article history:
Available online 3 March 2020

Keywords:
Asynchronous cellular automata
Non-camouflage
Freezing
Boolean totalistic
Turing machine
Elementary cellular automata

ABSTRACT

Asynchronous Boolean totalistic cellular automata have recently attracted attention as promising models for implementation by reaction-diffusion systems. It is unknown, however, to what extent they are able to conduct computation. In this paper, we introduce the so-called *non-camouflage property*, which means that a cell's update is insensitive to neighboring states that equal its own state. This property subsumes the *Boolean totalistic property*, which signifies the existence of states in a cell's neighborhood, but is not concerned with how many cells are in those states. We argue that the non-camouflage property is extremely useful for the implementation of reaction-diffusion systems, and we construct an asynchronous cellular automaton with this property that is Turing-complete by directly simulating Turing machines. We also construct another asynchronous cellular automaton, but this model incorporates the so-called *freezing* property [1], which restricts the direction of transitions of each cell to one-way. We show that this model is Turing-complete, since it can simulate the temporal evolution of elementary cellular automata. These results indicate the feasibility of computation by reaction-diffusion systems.

1. Introduction

Recent efforts towards the molecular implementation of reaction-diffusion systems have resulted in the characterization of cellular automata that are suitable for this purpose [2,3]. A possible implementation for this kind of CA uses a porous material, such as an *alginate* or *polyacrylamide gel*, as the framework of the cellular space. In this type of material, many small (millimeter scale) holes are arranged as a lattice, each of which is employed as a cell, with boundaries made of this material. Artificial DNA molecules are then used to represent cell states, whereby their chemical reactions represent a transition rule acting upon these states. These DNA molecules are broadly divided into two types according to their size. Small molecules are able to pass through the porous material at a cell's boundary, but big molecules are not. Big molecules are thus suitable to be used for representing the state of a cell, whereas small molecules can act as transmitters

* Corresponding author.
E-mail addresses: t.yamashita1238@gmail.com (T. Yamashita), isokawa@eng.u-hyogo.ac.jp (T. Isokawa), peper@nict.go.jp (F. Peper), kawamata@molbot.mech.tohoku.ac.jp (I. Kawamata), hagiya@is.s.u-tokyo.ac.jp (M. Hagiya).

https://doi.org/10.1016/j.ic.2020.104539

to neighboring cells. The reactions between molecules are designed according to the transition rule of the implemented CA. A computation on the CA is then initiated by injecting the designed molecules into each cell (hole) depending on the initial state of the CA. Computational cellular systems created by the above procedure are called *Cellular Automata* [4,5].

In the scheme outlined above, the implemented CA must satisfy certain requirements to allow it to exploit the characteristics of molecular implementations. Since it is difficult to synchronize the chemical reactions in all cells, the CA should be asynchronous, rather than an ordinary synchronous CA. In addition, it is also difficult for reaction-diffusion systems to recognize the direction from which DNA molecules have come, so, rather than identifying the state of each neighboring cell, we merely use the number of neighboring cells in certain states (totalistic CA). This is not sufficient, though, since it is quite difficult to estimate the amount of diffused DNA molecules in cells, and even to establish how many neighboring cells are in a certain state. For this reason, it was proposed to refine the totalistic CA to so-called *Boolean totalistic CA* [2], in which the mere presence and absence of states among the neighbors of a cell are sufficient in the definition of a transition rule.

There is an additional difficulty in this scheme, however. Imagine that a cell in a certain state is supposed to change to another state if there is a neighboring cell whose state is identical to the current state of the cell. Such a transition rule is allowed in a conventional asynchronous Boolean totalistic CA. However, in a reaction-diffusion implementation, a cell cannot recognize the existence of a neighboring cell in the same state since the cell itself is emitting the transmitter indicating its state. To resolve this difficulty, we define the *non-camouflage property* in this paper, which in effect ignores a state of a cell's neighbor if the state equals the state of the cell itself. This property subsumes the Boolean totalistic property. We present an asynchronous non-camouflage CA and prove that it is Turing-complete by simulating circuits that form Turing machines on the CA.

A second CA we construct is inspired by the notion that, without an external supply of substance and energy, a reaction-diffusion system cannot make transitions infinitely many times. This means cells have a limit on the number of possible transitions. We incorporate the so-called *freezing property* [1], which prohibits a cell from returning to an original state. We present an asynchronous non-camouflage freezing CA and show that it is also Turing-complete as it can simulate temporal evolution of elementary cellular automata.

This paper is organized as follows. Section 2 gives the formal definitions of the used concepts. This is followed by descriptions of the proposed CAs in Section 3 and Section 4. The two CA models are proven to be Turing-complete in Section 5. This paper finishes with a discussion in Section 6.

2. Preliminaries

2.1. Asynchronous CA

In this paper, we follow the terminology used in [6]. State transition systems, which are pairs of a set and a binary relation on the set, are called *state-systems*. We then define synchronous and asynchronous CA as state-systems. Note that ordinary CA are synchronous CA, while in this paper we only deal with asynchronous CA [7–10].

Definition 1 *(State-system).* A state-system A is a pair $A = (T, \to)$, where T is a set of states, and $\to \in T \times T$ is a binary relation meaning state transition.

For $(t_1, t_2) \in \to$, we write $t_1 \to t_2$ and say "the state t_1 is changed to t_2." Let $t_0, t_n \in T$ be states. If there are states $t_1, \ldots, t_{n-1}$ and $t_i \to t_{i+1}$ holds for each $i = 0, \ldots, n-1$, we write $t_0 \to^* t_n$.

To prove that a state-system B is computationally more powerful than a state-system A or equally powerful as A, we need to show that B can simulate A. Here is the definition of simulation derived from [6] but slightly modified for our purpose.

Definition 2 *(Simulation).* A state-system $B = (T_B, \to_B)$ simulates a state-system $A = (T_A, \to_A)$ if there is a function $F : T_A \to T_B$ and

(i) $\forall t_1, t_2 \in T_A.\ t_1 \to_A t_2 \Longrightarrow \forall t' \in T_B.\ F(t_1) \rightsquigarrow_F t' \Longrightarrow t' \rightsquigarrow_F F(t_2)$
(ii) $\forall t_1, t_2 \in T_A.\ F(t_1) \to_B^* F(t_2) \Longrightarrow t_1 \to_A^* t_2$

where $\rightsquigarrow_F$ denotes the binary relation over T_B that is defined as

$$t' \rightsquigarrow_F t'' \Longleftrightarrow \exists n \in \mathbb{N}.\ \exists t'_0, \ldots, t'_n \in T_B.$$
$$t' = t'_0 \to_B t'_1 \to_B \cdots \to_B t'_n = t'' \wedge \forall i \in [1, n-1].\ t'_i \notin F(T_A).$$

The function F in this definition is called a *simulation function* (or simply a *simulation*) of A by B.

Intuitively, (i) means that for any transition $t_1 \to_A t_2$, there is a sequence of transitions from $F(t_1)$ to $F(t_2)$, which does not go through the construction of A by F. Since the binary relation $\rightsquigarrow_F$ is reflexive, (i) implies

$$\forall t_1, t_2 \in T_A.\, t_1 \to_A t_2 \Longrightarrow F(t_1) \to_B^* F(t_2).$$

This corresponds to one of the original conditions of simulation in [6]. (ii) means that any sequence of transitions from $F(t_1)$ to $F(t_2)$ in B corresponds to a sequence of transitions from t_1 to t_2 in A.

Next, we define a CA as a state-system whose states can be considered as an arrangement in a certain topology[1] of cell states, whereby the transition relation is determined by applying a local transition rule to each cell simultaneously [11–13]. We define asynchronous CA, which are discussed mainly in this paper.

Definition 3 *(Asynchronous CA).* A state-system $A = (T, \to)$ is called an asynchronous CA if the following two conditions are satisfied.

(i) There is a set S_A of cell states such that $T = S_A^{\mathbb{Z}\times\mathbb{Z}}$.
(ii) There is a function $f_A : S_A^5 \to S_A$ such that for any two states $t_1, t_2 \in T$,

$$\begin{aligned} t_1 \to t_2 \Longleftrightarrow \forall x, y \in \mathbb{Z}.\, (t_2(x, y) = f_A(t_1(x, y), t_1(x+1, y), t_1(x, y+1), \\ t_1(x-1, y), t_1(x, y-1)) \\ \vee\, t_2(x, y) = t_1(x, y)). \end{aligned}$$

For an asynchronous CA A, S_A and f_A are called the *space of cell states* and the *transition rule*, respectively. An element of T is called a *configuration*.

2.2. Requirements

Here, we define the requirements for asynchronous CA to be implemented by reaction-diffusion systems. The following definitions are about asynchronous CA but the word "outer totalistic" can also be applied to synchronous CA.

Definition 4 *(Outer totalistic).* The function $tot : S_A^5 \to S_A \times \mathbb{N}^{S_A}$ is defined by $tot(s_0, s_1, s_2, s_3, s_4) = (s_0, h)$, where

$$h(s) = \sum_{k=1}^{4} g(s_k, s),\ g(s, s') = \begin{cases} 1 & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases}$$

are functions $h : S_A \to \mathbb{N}$ and $g : S_A \times S_A \to \mathbb{N}$. An asynchronous CA A is (outer) totalistic if there is a function $f'_A : S_A \times \mathbb{N}^{S_A} \to S_A$ and $f_A = f'_A \circ tot$.

Definition 5 *(Boolean totalistic).* Let the symbol 2 denote the set of Boolean values. The function $bol : S_A^5 \to S_A \times 2^{S_A}$ is defined by $bol(s_0, s_1, s_2, s_3, s_4) = (s_0, h)$, where

$$h(s) = \bigvee_{k=1}^{4} (s_k = s)$$

is a function $h : S_A \to 2$. An asynchronous CA A is Boolean totalistic if there is a function $f''_A : S_A \times 2^{S_A} \to S_A$ and $f_A = f''_A \circ bol$.

If an asynchronous CA A is Boolean totalistic, the transition rule is determined by the function $f''_A : S_A \times 2^{S_A} \to S_A$. We identify the transition rule f_A with f''_A and represent it by a list of the form like $s_0(s_i, \dots, \neg s_j, \dots) \to s'_0$. This expression means that a cell with state s_0 can be changed to state s'_0 when in its neighborhood there are cells with state $s_i, \dots$ and there is no cell with state $s_j, \dots$. We call such expressions *clauses*. If there is no clause whose left-hand side is applicable to a situation $(s_0, \dots, s_4)$, then f''_A returns s_0. A cell in such a situation $(s_0, \dots, s_4)$ will not change its state by a transition of A. If there is more than one clause whose left-hand side is applicable to a situation $(s_0, \dots, s_4)$, then f''_A follows the first one.

Since there is a function $g : S_A \times \mathbb{N}^{S_A} \to S_A \times 2^{S_A}$ such that $bol = g \circ tot$, an asynchronous Boolean totalistic CA is outer totalistic. In other words, the Boolean totalistic property subsumes the outer totalistic property.

As discussed in Section 1, the Boolean totalistic property is still not sufficient for our purposes. We define a non-camouflage property, which subsumes the Boolean totalistic property.

[1] We discuss CA with the two-dimensional lattice arrangement using the von Neumann neighborhood.

Definition 6 *(Non-camouflage).* An asynchronous Boolean totalistic CA A is non-camouflage if its transition rule f''_A satisfies

$$\forall s_0 \in S_A.\ \forall h_0, h_1 \in 2^{S_A}.$$
$$(\forall s \in S_A.\ s \neq s_0 \Longrightarrow h_0(s) = h_1(s)) \Longrightarrow f''_A(s_0, h_0) = f''_A(s_0, h_1),$$

If an asynchronous Boolean totalistic CA is non-camouflage, it is called an asynchronous non-camouflage CA.

Furthermore, to bound the transition times of cells, we define freezing CA. Once a cell in freezing CA changes its state, it cannot return to the original state. To define this property, we prepare a reachability graph.

Definition 7 *(Reachability graph).* The reachability graph of the CA A is a directed graph $\mathcal{G} = (V, E)$, where

- $V = S_A$ is the set of vertices;
- $E = \{(q, q') \in S_A \times S_A \mid \exists n \in S_A^4.\ q' = f_A(q, n)\}$ is the set of edges.

If a cell in state $q \in S_A$ can make a transition to state $q' \in S_A$, the reachability graph of A has an edge from q to q'.

Definition 8 *(Freezing).* A CA A is freezing iff the reachability graph of A is a directed acyclic graph (DAG).

It is easy to show that any cell in a freezing CA A cannot make transitions infinitely many times because the set of states S_A is finite.

2.3. Priese System

In Section 3 we construct an asynchronous non-camouflage CA. In [6], Priese defined a Turing-complete system, which we call a *Priese System* in this paper and define in this subsection. Note that a Priese System is suitable for our purposes because it does not require synchronization of its elements.

First, we define *s-automata* (named after sequential automata).

Definition 9 *(s-automaton).* A tuple $A = (I, O, S, \rightarrow)$ is called an s-automaton if

(i) I and O are finite sets with $I \cap O = \emptyset$.
(ii) S is a set.
(iii) $\rightarrow \subset (I \times S) \times (O \times S)$ is a transition relation.

An s-automaton $A = (I, O, S, \rightarrow)$ is thus a machine that has a set I of input terminals, a set O of output terminals, and a set S of inner states. Let $x \in I$, $y \in O$ and $s, s' \in S$ be an input terminal, an output terminal and inner states, respectively. The transition relation $((x, s), (y, s')) \in \rightarrow$ is denoted as $(x, s) \rightarrow (y, s')$. This state transition is interpreted as follows. If the input terminal x receives a signal and the s-automaton A is in the inner state s, then A can remove the signal on the input terminal x, change its inner state from s to s' and add a signal to the output terminal y. An s-automaton $A = (I, O, S, \rightarrow)$ can be considered as a state-system $((I \sqcup O) \times S, \rightarrow)$.[2]

Two s-automata called K and E are used to define the Priese System. The s-automaton $\mathrm{K} = (I_\mathrm{K}, O_\mathrm{K}, S_\mathrm{K}, \rightarrow_\mathrm{K})$ is defined by

$$I_\mathrm{K} = \{0, 1\},\ O_\mathrm{K} = \{2\},\ S_\mathrm{K} = \{0\},\ \rightarrow_\mathrm{K} = \{((0, 0), (2, 0)), ((1, 0), (2, 0))\}.$$

This s-automaton has two input terminals. Whichever input terminal receives a signal, it flows to the unique output terminal. The s-automaton $\mathrm{E} = (I_\mathrm{E}, O_\mathrm{E}, S_\mathrm{E}, \rightarrow_\mathrm{E})$ is defined by

$$I_\mathrm{E} = \{s, t\},\ O_\mathrm{E} = \{s', t^u, t^d\},\ S_\mathrm{E} = \{u, d\},$$
$$\rightarrow_\mathrm{E} = \{((s, u), (s', d)), ((s, d), (s', u)), ((t, u), (t^u, u)), ((t, d), (t^d, d))\}.$$

This s-automaton has two inner states. When a signal arrives at the input terminal t, it flows to the output terminal t^u or t^d depending on the inner state of the s-automaton E. When a signal arrives at the input terminal s, it flows to the output terminal s' and the inner state is flipped at the same time.

Both of the s-automata K and E are too simple to simulate a universal Turing machine on their own, but a system constructed by connecting them turns out to be powerful enough. To connect s-automata to each other, we define two operations over s-automata, i.e., product and feed-back.

The *product* of s-automata A and B is the s-automaton given by arranging them in a parallel configuration.

[2] $\sqcup$ denotes a disjoint union.

Definition 10 *(Product).* Let s-automata $A = (I_A, O_A, S_A, \to_A)$, $B = (I_B, O_B, S_B, \to_B)$ be given. The product $A \otimes B$ is the s-automaton $(I_A \sqcup I_B, O_A \sqcup O_B, S_A \times S_B, \to_{A\otimes B})$ with $\to_{A\otimes B} = \{((x,(s,t)),\ (y,(s',t))) \mid (x,s) \to_A (y,s'), t \in S_B\} \cup \{((x,(s,t)),(y,(s,t'))) \mid (x,t) \to_B (y,t'), s \in S_A\}$.

Let s-automaton A be given. The *feed-back* of the output terminal y to the input terminal x is the s-automaton given by connecting y to x.

Definition 11 *(Feed-back).* Let an s-automaton $A = (I_A, O_A, S_A, \to_A)$, an input terminal $x \in I_A$, and an output terminal $y \in O_A$ be given. The feed-back A^x_y of the output terminal y to the input terminal x is the s-automaton $(I_A \setminus \{x\}, O_A \setminus \{y\}, S_A, \to_{A^x_y})$ with

$$\to_{A^x_y} = Cl(\to_A \cup \{((y,s),(x,s)) | s \in S_A\}) \cap ((I_A \setminus \{x\} \times S_A) \times (O_A \setminus \{y\} \times S_A)),$$

where Cl denotes the transitive and reflexive closure of a binary relation.

If one wants to make a machine that is made of s-automata $A_1, \ldots, A_n$, one can put them in parallel by the operation product and connect them to each other by the feed-back operation. The class of s-automata generated by such operations is called *Normed Networks*.

Definition 12 *(Normed Network).* Let s-automata $A_1, \ldots, A_n$ be given. The Normed Network over $A_1, \ldots, A_n$ is the smallest set of s-automata that

(i) contains $A_1, \ldots, A_n$ and
(ii) is closed under feed-back and product.

A Priese System is defined as a Normed Network over s-automata K and E. It is known that any finite-state s-automaton belongs to a Priese System [6]. Next we define the infinite chain made of two s-automata A and B, where A and an infinite number of copies of B are connected in sequence.

Definition 13 *(Infinite chain).* Let s-automata $A = (I_A \sqcup \bar{I}_A, O_A \sqcup \bar{O}_A, S_A, \to_A)$ and $B = (I_B \sqcup \bar{I}_B, O_B \sqcup \bar{O}_B, S_B, \to_B)$ be given and $|\bar{I}_A| = |O_B| = |\bar{I}_B| = m$, $|\bar{O}_A| = |\bar{O}_B| = |I_B| = n$. Let $B^{(i)} = (I^{(i)}_B \sqcup \bar{I}^{(i)}_B, O^{(i)}_B \sqcup \bar{O}^{(i)}_B, S^{(i)}_B, \to^{(i)}_B)$ be disjoint copies of B and $\bar{I}_A = \{\bar{x}_1, \ldots, \bar{x}_m\}$, $\bar{O}_A = \{\bar{y}_1, \ldots, \bar{y}_n\}$, $I^{(i)}_B = \{x^{(i)}_1, \ldots, x^{(i)}_n\}$, $O^{(i)}_B = \{y^{(i)}_1, \ldots, y^{(i)}_m\}$, $\bar{I}^{(i)}_B = \{\bar{x}^{(i)}_1, \ldots, \bar{x}^{(i)}_m\}$, $\bar{O}^{(i)}_B = \{\bar{y}^{(i)}_1, \ldots, \bar{y}^{(i)}_n\}$. The infinite chain made of A and B is an s-automaton $(S, I, O, \to)$ where $S = \{(s, t_0, t_1, \ldots) \mid s \in S_A, t_i \in S^{(i)}_B\}$, $I = I_A$, $O = O_A$, and

$$\begin{aligned}
\to = Cl(&\{((x,(s,t_0,t_1,\ldots)),(y,(s',t_0,t_1,\ldots))) \mid (x,s) \to_A (y,s')\} \\
&\cup \{((x,(s,t_0,\ldots,t_i,\ldots)),(y,(s,t_0,\ldots,t'_i,\ldots))) \mid (x,t_i) \to^{(i)}_B (y,t'_i)\} \\
&\cup \{((\bar{y}_k,u),(x^{(0)}_k,u)) \mid \bar{y}_k \in \bar{O}_A, x^{(0)}_k \in I^{(0)}_B\} \\
&\cup \{((y^{(0)}_k,u),(\bar{x}_k,u)) \mid y^{(0)}_k \in O^{(0)}_B, \bar{x}_k \in \bar{I}_A\} \\
&\cup \{((\bar{y}^{(i)}_k,u),(x^{(i+1)}_k,u)) \mid \bar{y}^{(i)}_k \in \bar{O}^{(i)}_B, x^{(i+1)}_k \in I^{(i+1)}_B\} \\
&\cup \{((y^{(i+1)}_k,u),(\bar{x}^{(i)}_k,u)) \mid y^{(i+1)}_k \in O^{(i+1)}_B, \bar{x}^{(i)}_k \in \bar{I}^{(i)}_B\}) \\
\cap ((I \times S) &\times (O \times S)).
\end{aligned}$$

It is known that any computation of a Turing machine from a finite initial configuration can be simulated by a computation of an infinite chain of finite-state s-automata. An argument showing this fact can be found, for instance, in [14].

3. Asynchronous non-camouflage CA

3.1. Proposed CA M

In this subsection, we present an asynchronous non-camouflage CA on a 2-dimensional lattice.

Table 1 shows the transition rule of our asynchronous non-camouflage CA in the asynchronous Boolean totalistic form. For simplicity, we call this asynchronous CA M. By the definition of asynchronous CA, M is a state system $\mathrm{M} = (S_{\mathrm{M}}^{\mathbb{Z}\times\mathbb{Z}}, \to_{\mathrm{M}})$. The number of cell states becomes $|S_{\mathrm{M}}| = 21$ by adding a quiescent state 0 to the 20 states appearing in the table of the transition rule. All of state symbols in M with their functionalities are shown in Table 2.

In the table of the transition rule, state s_0 of each clause does not appear in the bracket $(s_i, \ldots, \neg s_j, \ldots)$. This fact implies that M is non-camouflage.

Table 1
Transition rule of M.

1	$1\ (2, \neg C_1, \neg E_0) \to Z$	13	$Z\ (3, U_0, \neg L, \neg D_0, \neg E_1) \to u$
2	$1\ (W, \neg K) \to Z$	14	$Z\ (3, D_0, \neg L, \neg U_0, \neg E_1) \to d$
3	$1\ (u, \neg E_0) \to Z$	15	$C_0\ (Z) \to C_1$
4	$1\ (d, E_0) \to Z$	16	$C_1\ (4) \to C_0$
5	$2\ (3, Z, \neg U_0, \neg D_0) \to Y$	17	$W\ (3, Z) \to Y$
6	$3\ (4, Y) \to X$	18	$U_0\ (1, 2, E_0) \to D_1$
7	$4\ (X, \neg C_1, \neg U_1, \neg D_1) \to 1$	19	$D_1\ (1, 4, E_0) \to D_0$
8	$X\ (1, Y, \neg 4) \to 4$	20	$D_0\ (1, 2, E_0) \to U_1$
9	$Y\ (4, Z) \to 3$	21	$U_1\ (1, 4, E_0) \to U_0$
10	$Z\ (3, \neg C_0, \neg L, \neg D_0, \neg U_0) \to 2$	22	$u\ (3, Z) \to Y$
11	$Z\ (3, L) \to W$	23	$d\ (3, Z) \to Y$
12	$Z\ (3, E_1, \neg L) \to 2$		

Table 2
List of 21 states in M with their functionalities.

State symbol(s)	Functionality
0	quiescent state
1	wire state
$2, 3, 4$	states for a signal (2: head of signal and 4: tail of signal)
X, Y, Z	intermediate states for a signal going ahead
C_0, C_1	input/output terminals for crossing element (C_0: terminal waiting for a signal coming, C_1: terminal processing a signal)
K, L	terminal states for K-element (K: input terminal, L: output terminal)
W	intermediate state for signal in K-element
E_0, E_1	terminal states for E-element (E_0: terminal for input T, E_1: terminal for input S)
D_0, U_0	states for E-element (D_0: in state d, U_0: in state u)
D_1, U_1	intermediate states when changing the state in E-element for an input S
d, u	intermediate states when routing a signal in E-element for an input T

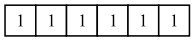

Fig. 1. Wire on M.

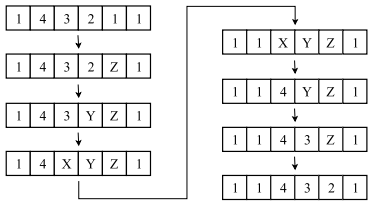

Fig. 2. Signal on a wire on M.

Since a cell in the state 0 will never be changed to another state or influence transitions of neighboring cells, such cells are not drawn in the figure. Almost all cells are in the cell state 0.

3.2. Simulation of s-automaton

In this subsection, we show that M simulates any s-automaton belonging to the Priese System. M can also simulate an infinite chain of which the elements consist of two s-automata in a Priese System with an initial configuration that is periodic except for a finite area. The universality of M follows from this fact.

3.2.1. Wires and signals

Cells with the state $1 \in S_{\mathrm{M}}$ extending linearly are called a wire. Fig. 1 shows a construction of a wire. Three cells in the states $2, 3, 4 \in S_{\mathrm{M}}$ arranged in this order are called a signal. This order makes a signal directed. A signal on a wire progresses along the wire as in Fig. 2. A signal on a finite wire reaches the end of the wire in a finite number of transitions if there is no influence from the outside on the wire.

Let $A = (I_A, O_A, S_A, \to_A)$ be an s-automaton. Recall that A is simulated as a state-system $((I_A \sqcup O_A) \times S_A, \to_A)$. A state $(io, s) \in (I_A \sqcup O_A) \times S_A$ of A is regarded as a situation in which a machine with an inner state s has a signal on a terminal io. Wire-based simulation functions are simulation functions of s-automata by M, and they represent such situations. Fig. 3 shows how a wire-based simulation represents a transition $(x, s) \to (y, s')$ of an s-automaton with one input terminal x and one output terminal y.

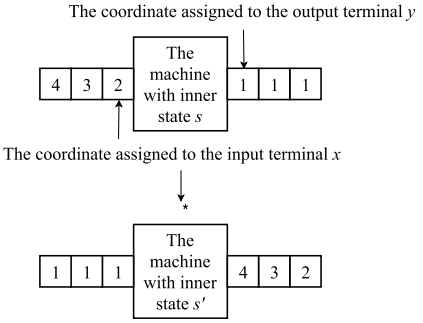

Fig. 3. Construction of wire-based simulation.

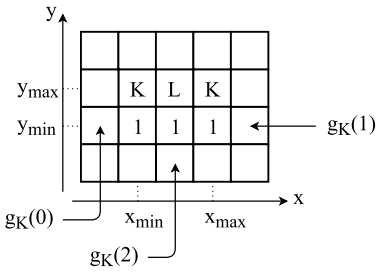

Fig. 4. $G_K(0)$ and g_K on M.

Definition 14 *(Wire-based simulation).* Let $A = (I_A, O_A, S_A, \rightarrow_A)$ be an s-automaton and $F : (I_A \sqcup O_A) \times S_A \rightarrow S_M^{\mathbb{Z}\times\mathbb{Z}}$ be a simulation function of A by M. F is called a wire-based simulation function if there are functions $G : S_A \rightarrow S_M^{\mathbb{Z}\times\mathbb{Z}}$, $g : I_A \sqcup O_A \rightarrow \mathbb{Z} \times \mathbb{Z}$ and $x_{min}, x_{max}, y_{min}, y_{max} \in \mathbb{Z} \cup \{-\infty, \infty\}$ such that

(i) for each state $s \in S_A$ of A, there is no transition from $G(s)$,
(ii) $\forall s \in S_A. \forall x, y \in \mathbb{Z}. G(s)(x, y) = 0 \vee (x_{min} \leq x \leq x_{max} \wedge y_{min} \leq y \leq y_{max})$,
(iii) $\forall io \in I_A \sqcup O_A. \forall x, y \in \mathbb{Z}. g(io) = (x, y) \Longrightarrow ((x = x_{min} - 1 \vee x = x_{max} + 1) \wedge (y_{min} < y < y_{max})) \vee ((y = y_{min} - 1 \vee y = y_{max} + 1) \wedge (x_{min} < x < x_{max}))$,
(iv) for each input terminal $i \in I_A$ and state $s \in S_A$ of A, the state $F(i, s)$ is identical with the state constructed by replacing a series of three cells with states $(0, 0, 0)$ in $G(s)$, which is extended toward the outside from the coordinate $g(i)$, with an inwardly directed signal $(2, 3, 4)$, and replacing series of three cells with states $(0, 0, 0)$, which are extended toward the outside from the coordinate $g(io)$ for each terminal $io \in (I_A \setminus \{i\}) \sqcup O_A$, with wires $(1, 1, 1)$, and
(v) for each output terminal $o \in O_A$ and state $s \in S_A$ of A, there is no difference between the procedure of constructing the state $F(o, s)$ and (iv) but placing the signal in the opposite direction.

In this definition, the rectangle region $\{(x, y) \in \mathbb{Z} \times \mathbb{Z} \mid x_{min} \leq x \leq x_{max} \wedge y_{min} \leq y \leq y_{max}\}$ is called the *frame* of F.

Note that if functions G and g are given, the wire-based simulation function F is uniquely determined by the conditions (iv) and (v) of Definition 14. So we regard the pair of G and g as F. In the figures, the function $G(s)$ for a state $s \in S_A$ is shown as an arrangement of cell states, and the function $g(io)$ is shown by an arrow pointing at the cell corresponding to the input/output terminal io.

3.2.2. Simulation of K

We construct a wire-based simulation of the s-automaton K now. Fig. 4 shows the state $G_K(0)$ and the coordinate $g_K(io)$ for each terminals $io \in I_K \sqcup O_K$. They give wire-based simulation F_K of the s-automaton K in state $0 \in S_K$. Fig. 5 shows the state $F_K(0, 0)$. The states simulating K with a signal in the other input/output terminals are also constructed in the same way.

Now we confirm that the function F_K determined by Fig. 4 and the conditions of Definition 14 is a wire-based simulation function of K. Since there is no clause in Table 1 that is applicable to a cell in $G_K(0)$, the state $G_K(0)$ cannot be changed. Thus, the condition (i) of Definition 14 is satisfied. Recall that the cells which are not drawn in figures are assumed to be in state $0 \in S_M$. That means the condition (ii) of Definition 14 is satisfied if the frame of F_K is determined by $x_{min}, x_{max}, y_{min}$ and y_{max} in Fig. 4. The condition (iii) of Definition 14 is also satisfied because of the way to interpret the figure. The conditions (iv) and (v) are satisfied because the function F_K is constructed by these conditions. Thus, if the function F_K is a simulation function of K, F_K is a wire-based simulation function.

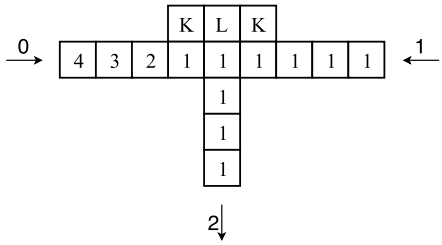

Fig. 5. $F_{\mathrm{K}}(0,0)$ on M.

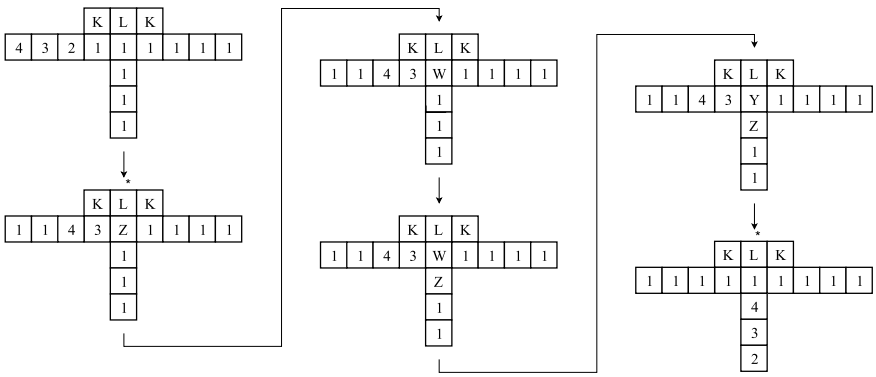

Fig. 6. Transitions from $F_{\mathrm{K}}(0,0)$ to $F_{\mathrm{K}}(2,0)$ on M.

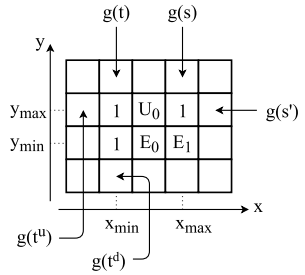

Fig. 7. $G_{\mathrm{E}}(u)$ on M.

Fig. 6 shows the transitions that can be made if the initial state is $F_{\mathrm{K}}(0,0)$. Multiples of transitions are presented by the symbols $\rightarrow^*$ to save space. These omitted transitions are almost the same as the transitions shown in Fig. 2. Any state t' which satisfies $F_{\mathrm{K}}(0,0) \rightsquigarrow_{F_{\mathrm{K}}} t'$ appears in the transitions shown in Fig. 6, and any state t' in these transitions satisfies $t' \rightsquigarrow_{F_{\mathrm{K}}} F_{\mathrm{K}}(2,0)$. A similar argument holds for transition $(1,0) \rightarrow_{\mathrm{K}} (2,0)$, so the condition (i) of Definition 2 is satisfied.

Since there is no transition of M to the states $F_{\mathrm{K}}(0,0)$ or $F_{\mathrm{K}}(1,0)$ and there is no transition of M from the state $F_{\mathrm{K}}(2,0)$, the condition (ii) of Definition 2 is also satisfied.

Therefore the function F_{K} is a simulation function of K.

3.2.3. *Simulation of* E

The s-automaton E with the inner state u and, alternatively, inner state d is simulated by the arrangement of cell states shown in Fig. 7 and Fig. 8, respectively. The unique difference between these two states is whether the state of the center cell is in state U_0 or in state D_0. As in the case of K, we can confirm that the function F_{E} determined by Figs. 7 and 8 is a wire-based simulation of E.

The procedures to confirm that transitions starting from a state on M satisfy the conditions of Definition 2 are mechanical but complicated. In practice, we conducted these procedures by using a computer program, which simulates transitions on M and verified the conditions of Definition 2.

3.2.4. *Crossing on* M

Since we are dealing with a two-dimensional space, it is important to be able to connect pairs of terminals with wires so that they do not interfere with each other. We solve this problem by constructing a crossing, which allows two wires to cross each other. The construction of a crossing is shown in Fig. 9.

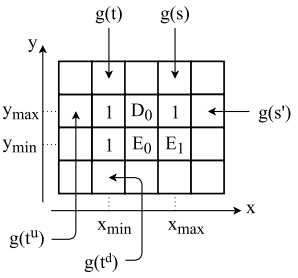

Fig. 8. $G_E(d)$ on M.

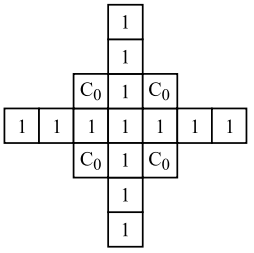

Fig. 9. Crossing on M.

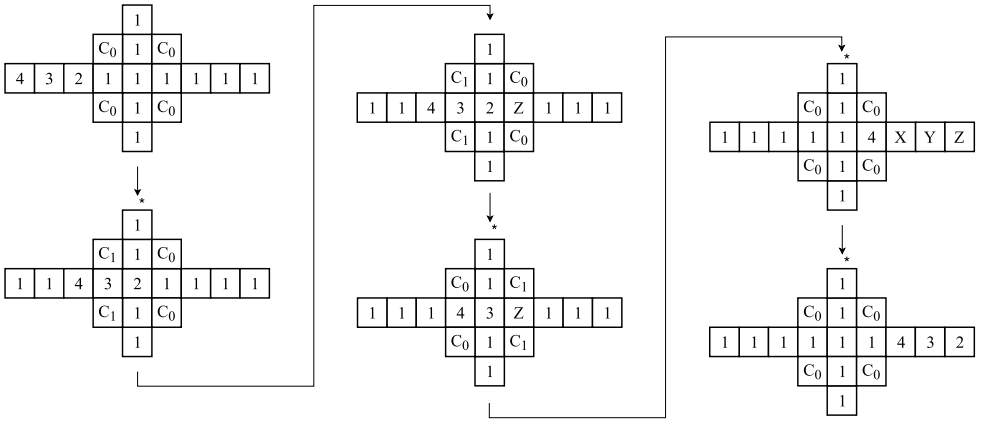

Fig. 10. Signal on a crossing on M.

Fig. 10 shows how a signal progresses across another wire. A wire cell in state 1 is usually changed to state Z when a cell in state 2 is in its neighborhood. However, a wire cell in contact with a cell in state C_1 is not changed because of clause 1 in Table 1. This is the reason why a signal progresses straight to the center of a crossing. These transitions are non-deterministic on the scale of individual cells, but in the end the crossing will have a signal at the opposite side of the signal in the crossing's initial state. Thanks to crossings, we can connect terminals freely.

3.2.5. Product and feed-back

We have proved that there are wire-based simulation functions of K and E. Next we will explain how to combine them.

Let $A = (I_A, O_A, S_A, \rightarrow_A)$ and $B = (I_B, O_B, S_B, \rightarrow_B)$ be s-automata. Assume that there are wire-based simulation functions F_A of A and F_B of B. Then, we can construct a wire-based simulation $F_{A\otimes B}$ of $A \otimes B$ by the following procedure.

First, we assume that all coordinates corresponding to terminals adjoin the right edge of the frame of the wire-based simulation functions. This assumption is possible because we can extend wires freely. Second, we construct $G_{A\otimes B}(s, s')$ for state $(s, s') \in S_A \times S_B$ by putting $G_A(s)$ and $G_B(s')$ in parallel vertically in such a way that the right sides are aligned. Then, $g_{A\otimes B}$ indicates the coordinates to which the wires have been extended in the first step. The function $F_{A\otimes B}$ determined by these $G_{A\otimes B}$ and $g_{A\otimes B}$ is a wire-based simulation of $A \otimes B$. The left part figure of Fig. 11 shows the construction of $G_{A\otimes B}(s, s')$.

Let $A = (I_A, O_A, S_A, \rightarrow_A)$, be an s-automaton, $x \in I_A$ its input terminal, and $y \in O_A$ its output terminal. Assume that there is a wire-based simulation function F_A of A. A wire-based simulation function $F_{A^x_y}$ of the feed-back A^x_y is constructed

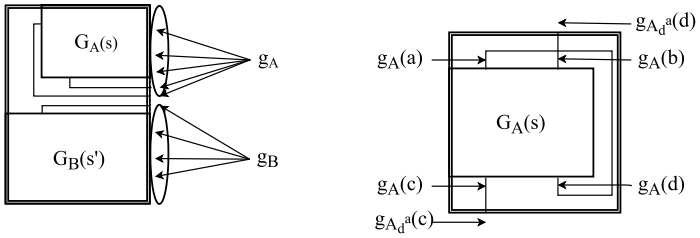

Fig. 11. Left: $G_{A\otimes B}(s, s')$. Right: $G_{A_d^a}(s)$. The frames of both constructions are represented by rectangles drawn with double lines.

Table 3
Transition rule of N.

1	$S_1\ (S_2) \to S_2$	7	$Z_Y\ (X_2) \to Z_{XY}$
2	$X\ (S_2) \to X_2$	8	$W\ (Z_X) \to S_2$
3	$Y\ (S_2) \to Y_2$	9	$W\ (Z_{XY}) \to S_2$
4	$Z\ (X_2) \to Z_X$	10	$V\ (Z_Y) \to S_2$
5	$Z\ (Y_2) \to Z_Y$	11	$V\ (Z_{XY}) \to S_2$
6	$Z_X\ (Y_2) \to Z_{XY}$	12	$R\ (X_2, Y_2) \to S_2$

as follows. First, a wire is extended from $g_A(y)$ to $g_A(x)$. Next, the frame of $F_{A_y^x}$ is installed so that it includes the whole frame of F_A and the wire extended in the first step. Finally, for each terminal $io \in (I_A \setminus \{x\}) \sqcup (O_A \setminus \{y\})$, a wire is extended outward from $g_A(io)$, and the coordinate $g_{A_y^x}(io)$ is determined by the end of the extended wire. A crossing is used if wires need to cross each other.

The right part figure of Fig. 11 shows a construction of the feed-back A_d^a. Two terminals a and d are connected by a wire. The connecting wire crosses the wire extended from $g_A(b)$, so a crossing must be used at this point.

3.2.6. Infinite chain

Let A and B be finite s-automata in a Priese System, and let F_A and F_B be the corresponding wire-based simulation functions. Assume that A and B satisfy the assumptions of Definition 13. A wire-based simulation of the infinite chain made of A and B can be constructed by putting G_A and an infinite number of copies of G_B in sequence and connecting the corresponding terminals to each other.

4. Asynchronous non-camouflage freezing CA

CAs with the hexagonal lattice are more suitable for modeling physical system [15], biological simulation [16,17], as well as molecular implementations of computation [3,18]. An asynchronous hexagonal totalistic CA with capability of universal computation has been proposed in [19].

In this section, we propose an asynchronous non-camouflage freezing CA on a hexagonal lattice. Let us call it N. In Section 2, we defined CAs and their properties on the square lattice, but it is easy to modify those definitions for the hexagonal one.

When a CA is implemented by a reaction-diffusion system, molecules acting as transmitters reach neighboring cells from a source cell by diffusion. Such transmitters are designed to travel a certain distance and then get decomposed. In a square lattice with von Neumann neighborhood, the transmitters at the corners can reach a neighboring cell that only shares a vertex with a source cell, and this may cause errors. On a hexagonal lattice, on the other hand, such errors are avoided, because two cells sharing a vertex also share an edge. A hexagonal lattice has thus advantages over a square one for implementing reaction-diffusion systems, which is a reason to define N on a hexagonal lattice.

We show that N can simulate any elementary CA (ECA). An ECA is a synchronous one-dimensional CA whose cells have only two states 0 and 1. Cook [20] showed that an ECA called “Rule 110” is Turing-complete.

4.1. Proposed CA N

Table 3 shows the transition rule of the asynchronous CA N in the Boolean totalistic form. Similar to M, $\mathrm{N} = (S_{\mathrm{N}}^{\mathbb{Z}\times\mathbb{Z}}, \to_{\mathrm{N}})$ is also a state system. We assume that cells arranged in the hexagonal lattice are indexed by $\mathbb{Z} \times \mathbb{Z}$. The number of states $|S_{\mathrm{N}}|$ is 14, adding an quiescent state to the 13 states appearing in Table 3.

Like with M, state s_0 of each clause does not appear in the bracket $(s_i, \ldots, \neg s_j, \ldots)$, implying that N is also non-camouflage.

The reachability graph of N is shown in Fig. 12. Note that the node of the quiescent state is omitted. Since this graph is a DAG, N is freezing.

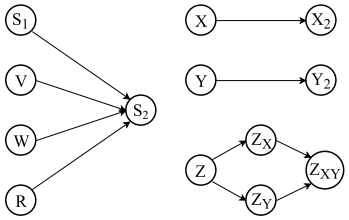

Fig. 12. Reachability graph of N.

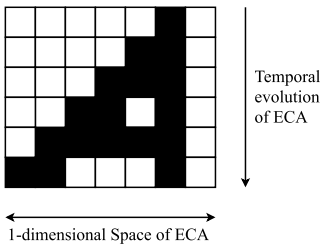

Fig. 13. Two-dimensional representation of temporal evolution of ECA.

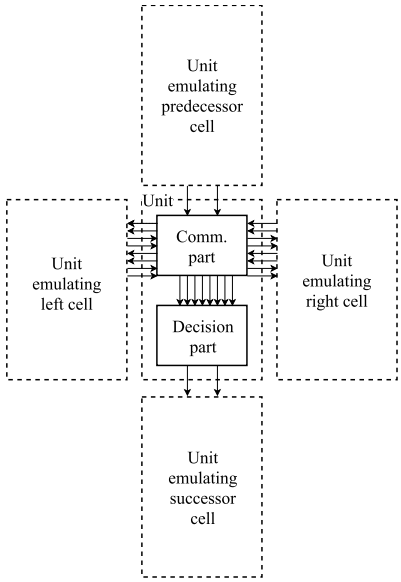

Fig. 14. Connection of units and their division into communication part and decision part.

4.2. Simulation of ECA

If CA is freezing, the cells cannot make transitions infinitely many times, unlike in Section 3, where K and E do not have any limit on the number of transitions. For this reason, we cannot directly simulate Turing machines by a freezing CA.

Rather, we embed the temporal evolution of an ECA in two-dimensional space, as shown in Fig. 13. We construct an s-automaton called a *unit*, which corresponds to a square in Fig. 13. Units horizontally connected with each other represent a state of the ECA, and they undergo transitions depending on the rule of the ECA.

Since a cell of the ECA undergoes a transition depending on the previous states of its neighbors and itself, a unit operates in two phases, i.e., it first obtains the states of the upper three units, and then determine the state it has to take, so the unit is divided into two s-automata, a *communication part* and a *decision part* as shown in Fig. 14.

4.3. Communication part

The communication part has 10 input terminals i_0, i_1, l_0, l_1, l_2, l_3, l_4, r_0, r_1, r_2, r_3, and 16 output terminals o_{000}, o_{001}, o_{010}, o_{011}, o_{100}, o_{101}, o_{110}, o_{111}, $\overline{l_0}$, $\overline{l_1}$, $\overline{l_2}$, $\overline{l_3}$, $\overline{r_0}$, $\overline{r_1}$, $\overline{r_2}$, $\overline{r_3}$, as shown in Fig. 15. The communication part receives an input from

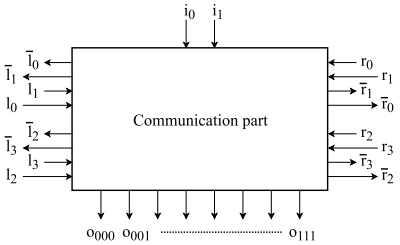

Fig. 15. Communication part.

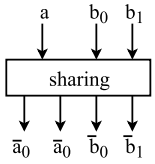

Fig. 16. Sharing element.

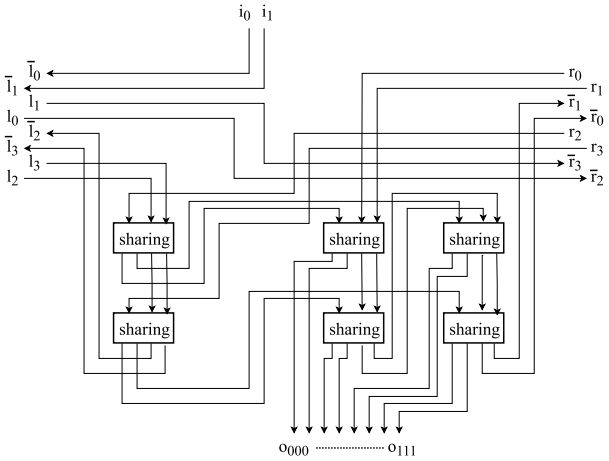

Fig. 17. Construction of communication part from sharing elements.

the decision part of the upper unit via terminals $i_{\{0,1\}}$. The input is a signal either from i_0 or from i_1. The communication part then forwards the input to the communication parts of the right and left units in turn via terminals $\overline{l_{\{0,1\}}}$, $l_{\{0,1\}}$, $\overline{r_{\{2,3\}}}$, and $r_{\{2,3\}}$. As we explain later, Fig. 18 depicts how the input travels among three communication parts. The input goes through several sharing elements, which are explained later. Each communication part reads an output from the neighboring communication parts via input terminals $l_{\{2,3\}}$ and $r_{\{0,1\}}$. Finally, it transfers 3-bit information via eight terminals as one-hot encoding to the lower decision part.

In order to share information among s-automata in general, we develop an s-automaton called *sharing element*. As shown in Fig. 16, a sharing element is an s-automaton which has four inner states $inactive$, $active_0$, $active_1$, and $spent$, three input terminals a, b_0, and b_1, four output terminals $\bar{a_0}$, $\bar{a_1}$, $\bar{b_0}$, and $\bar{b_1}$, with the transition relation defined as:

$$(a, active_i) \to (\bar{a_i}, spent)$$
$$(b_i, inactive) \to (\bar{b_i}, active_i).$$

When a signal reaches the input terminal b_0 while the state of the sharing element is $inactive$, it is switched to $active_0$ and then the signal is sent to the terminal $\bar{b_0}$. Similarly, a signal entering the other terminal b_1 exits terminal $\bar{b_1}$ while setting the state of the sharing element to $active_1$. When a signal reaches the input terminal a and the inner state is $active_i$, the signal goes to the output terminal $\bar{a_i}$. On the other hand, if there is a signal on the input terminal a while the inner state is $inactive$, the sharing element makes no transition and waits for a signal to come to the input terminal b_0 or b_1.

We can construct the communication part by using sharing elements as shown in Fig. 17.

We assume that only one of the two input terminals i_0 and i_1 of the communication part receives a signal depending on the current state of the unit that is considered as a cell of the ECA. As shown in Fig. 18, the signal then goes to

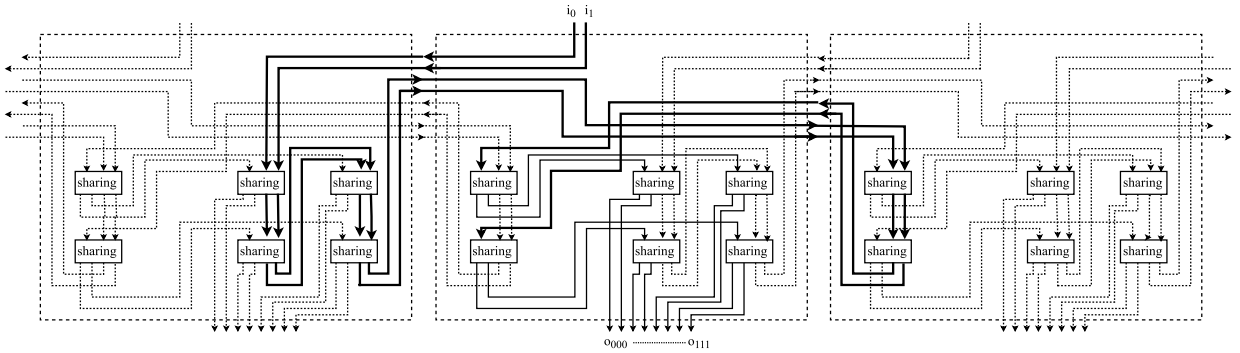

Fig. 18. Route of an input arriving at terminal i_0 or i_1.

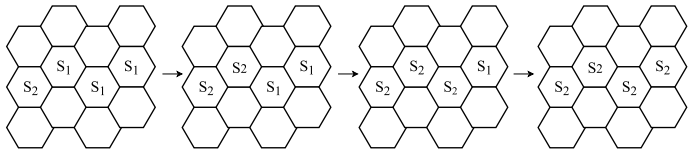

Fig. 19. Wire and signal on N.

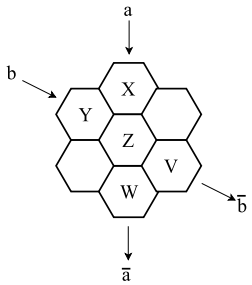

Fig. 20. Crossing element on N.

the corresponding terminal b_0 or b_1 of the sharing element inside the neighboring communication part in order to share the information of the current state. The signal finally goes to the corresponding terminal a of the sharing element inside the communication part itself. The communication parts hold the states of the neighboring units as the neighboring communication parts have also executed the same previous step. Pairs of wires emanating from the sharing elements then constitute the eight wires that form the output terminals of the communication part; these eight wires represent the output information by one-hot encoding.

The communication part described above is constructed from three kinds of elements: wires to control a signal, crossing elements to let two wires cross, and sharing elements.

4.3.1. Wire on N

A series of cells on N in state S_1 represents a wire, and a cell in state S_2 adjacent to a cell in state S_1 works as a signal. According to the first clause $S_1\ (S_2) \to S_2$ of Table 3, a signal makes progress along the adjacent wire, as shown in Fig. 19.

4.3.2. Crossing element on N

A crossing element has two input terminals a and b and two output terminals $\bar{a}$ and $\bar{b}$, as shown in Fig. 20. When a signal arrives at an input terminal, the signal goes to the corresponding output terminal.

When the signal is adjacent to the cell in state X, which corresponds to the input terminal a, a transition occurs from X to X_2. The state X_2 induces the central cell in state Z or Z_Y to make a transition to the state Z_X or Z_{XY}, respectively. These states tell the neighboring cells that the signal has come to input terminal a, after which the cell in state W, which corresponds to the output terminal $\overline{a}$, changes its state to S_2. This produces a signal on the corresponding output wire. The crossing element works similarly in the case that the signal comes to the input terminal b.

Note that the crossing element constructed here works even if both of the input terminals a and b receive signals simultaneously, because it processes the signals one by one in an arbitrary order. Thus, it can even be used to cross two wires of two distinct s-automata.

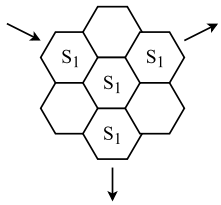

Fig. 21. Fan-out element on N.

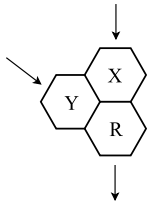

Fig. 22. Synchronizer element on N.

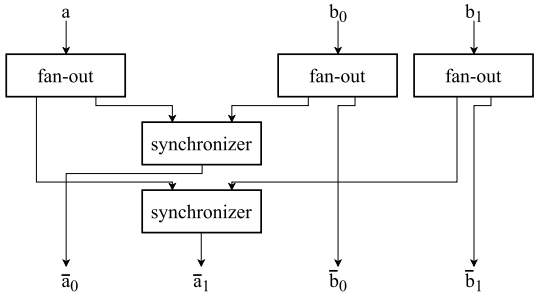

Fig. 23. Sharing element on N.

4.3.3. Sharing element

We have already mentioned the specification of a sharing element. Since it is connected with two distinct s-automata, it has to work independently of the order of two input signals.

To construct a sharing element, we prepare two elements called *fan-out element* and *synchronizer element*. Note that these are not s-automata because they may change the number of existing signals.

A fan-out element has an input terminal and two output terminals. When a signal arrives at the input terminal, this element sends signals to both of the output terminals. It is constructed simply by arranging cells in state S_1 in the shape of a 'Y' as shown in Fig. 21.

A synchronizer element has the inverse function of a fan-out element. It has two input terminals and one output terminal, and waits for signals to arrive at both of the input terminals. Once it receives two signals, it sends a signal to its unique output terminal. The pattern of cells realizing a synchronizer element is shown in Fig. 22. Through the application of the twelfth clause $R\,(X_2,\,Y_2) \to S_2$ in Table 3, cell R undergoes a transition after the two input terminals receive signals.

Now that we have defined the fan-out and synchronizer elements, we can construct a sharing element by combining the elements as in Fig. 23.

This completes the construction of the communication part on N.

4.4. Decision part

The decision part is easier to construct than the communication part. It receives the output of the communication part in the same unit. Each of the eight output terminals of the communication part indicates the state of the corresponding cell and its neighborhood. It then sends an output signal indicating the next state in accordance with the transition rule of the ECA.

To simulate the decision part, we introduce s-automata called *confluent elements*. A confluent element has two input terminals {0, 1}, one output terminal {2}, and two inner states {0, 1}. Its transition relation is defined as:

$(0,\,0) \to (2,\,1)$

$(1,\,0) \to (2,\,1)$.

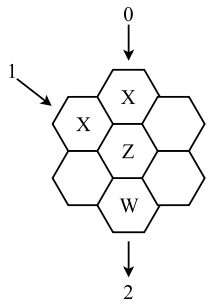

Fig. 24. Confluent element on N.

Thus, it sends a signal from either of the input terminals to the unique output terminal. This is similar to K of a Priese System, but differs in that it cannot be reused. Using confluent elements and crossing elements, the decision part can be constructed by connecting each input terminal with the corresponding output terminal in accordance with the transition rule of the ECA.

4.4.1. Confluent element on N

A confluent element can be simulated by the pattern of cells shown in Fig. 24. This construction uses the states also used in a crossing element. In the case of a crossing element, it was essential to distinguish between two inputs, so we arranged the state X at one side and the state Y at the other side, but in the case of a confluent element, the state X is arranged at both input terminals. The output terminal corresponding to state V is deleted because it is never be used.

Since a decision part can also be simulated on N, a whole unit can be simulated on N.

5. Turing-completeness

In this section we establish a closer connection between CA M and CA N via Turing completeness. Our argument revolves around the periodicity of the CAs beyond a finite area that is limited by a constant boundary, and it resembles Cook's construction, which shows that, given an initial configuration of rule 110 in an ECA that is eventually periodic to the left and right, it is RE-complete whether a given finite pattern ever occurs during the evolution.

Any Turing machine belonging to a reasonably restricted but still universal class of machines can be represented by an infinite chain of finite-state s-automata and some additional finite-state s-automaton as discussed in [14]. The chain represents the tape of the Turing machine, and the additional s-automaton represents the finite state control of the machine. We can define the entire s-automaton in such a way that the s-automaton produces an output if and only if the Turing machine halts.

5.1. Turing-completeness of CA M

As shown in Section 3, the entire s-automaton simulating the Turing machine can be simulated by CA M. The configuration of M representing the initial state of the entire s-automaton is *eventually periodic* in the following sense.

Definition 15 *(Eventually periodic).* A configuration $t \in T$ of an asynchronous CA $(T, \rightarrow)$ is called eventually periodic if there exist $m_1, m_2, m_3, m_4 > 0$, $p_1, p_2, p_3, p_4 > 0$ such that $t(i, j) = t(i + p_1, j)$ for $i > m_1$, $t(i, j) = t(i - p_1, j)$ for $i < -m_2$, $t(i, j) = t(i, j + p_3)$ for $j > m_3$, and $t(i, j) = t(i, j - p_4)$ for $j < -m_4$.

Although the finite part of the infinite chain representing a given input on the tape is not periodical in general, the rest of the infinite chain can be periodical.

The output produced by the s-automaton corresponds to a specific cell on M, which we call the *reporter* cell, and if the output is produced, there exists a transition sequence on M from its initial configuration to a configuration in which the reporter cell enters a certain state (possibly 2), which we call the *target* state. The converse also holds. If the reporter cell enters the target state, then the Turing machine halts.

Consequently, given an eventually periodic initial configuration, a cell and a cell state on M, it is undecidable whether the initial configuration can reach a configuration in which the cell is in the state, or not.

For testing the above reachability, any scheduling is allowed. We define the notion of *proper* transitions.

Definition 16 *(Proper transition).* For configurations t_1 and t_2, we call the transition from t_1 to t_2 proper if there exists at least one cell (x, y) such that $t_2(x, y)$ is determined by application of some clause in the transition rule.

For simulation of an s-automaton $A = (I, O, S, \rightarrow)$ by a wire-based simulation function F, we can verify the following property. If $(i, s) \rightarrow (o, s')$, any proper transition sequence starting from $F(x, s)$ eventually reaches $F(o, s')$.

Therefore, if the Turing machine halts, the reporter cell eventually enters the target state in any proper transition sequence from the initial configuration. In other words, there is no infinite proper transition sequence in which the reporter cell never enters the target state.

5.2. Turing-completeness of CA N

In the case of N, more complex arguments are needed for showing its Turing-completeness than in the case of M.

For the simulation of a cyclic tag system by 110 [20], two periodical patterns are used to the left and to the right on the one-dimensional cell space. Therefore, the initial configuration on N is eventually periodical.

The cyclic tag system uses a string of Y and N to represent a symbol in the tag system that simulates a Turing machine [20]. Therefore, if the Turing machine halts, the string of Y and N that corresponds to a certain symbol in the tag system, which in turn corresponds to the halting state of the Turing machine, is appended at the end of the tape of the cyclic tag system. This event appears as a pattern in a finite rectangular region on N. Therefore, by overlaying circuits consisting of fan-out, synchronizers and crossing elements, it is possible to detect the event. Furthermore, with a special new cell state and additional clauses in the transition rule, once the event is detected, the special state is generated and broadcast over the cell space.

Therefore, any cell can be chosen as the reporter cell, and the special state can be taken as the target state. The Turing machine halts if and only if the reporter cell enters the target state in some configuration that is reachable from the initial configuration.

In the case of N, scheduling should be taken into account since all cells on 110 should progress in a fair manner. This corresponds to considering infinite proper transition sequences in which each cell is made *active* infinitely often.

Definition 17 *(Active cell).* In a proper transition from t_1 to t_2, a cell (x, y) is called active if a clause in the transition rule is applied on it or no clause is applicable on it.

In other words, a cell is inactive if no clause in the transition rule is applied even though some clause is applicable.

If the Turing machine halts, then there is no infinite proper transition sequence in which the reporter cell never enters the target state even though each cell is made active infinitely often.

6. Discussion

This paper has presented two asynchronous non-camouflage CAs that are suitable for implementation by a reaction-diffusion system. One can directly simulate Turing machines and is more suitable for the implementation of molecular circuits, because it is based on a 2-dimensional lattice, and the other is freezing and based on a 2-dimensional hexagonal CA simulated on an ECA, which makes it more suitable for a reaction-diffusion system. The former has been obtained by making changes to the asynchronous totalistic CA presented in [2]. Both the asynchronous CAs presented in this paper are Turing-complete.

While a hexagonal lattice is most suitable for implementing N, it is not obvious to define M on a hexagonal lattice, though it is preferred to a square lattice for implementation by a reaction-diffusion system. Defining M on a hexagonal lattice is left as future work.

The transition rules of the asynchronous CAs proposed in this paper are represented by fewer clauses than those of the previous CA in [2]; the 23 clauses of the transition rule of CA M are less than half of the 57 clauses of the CA in [2]. This is a surprising result because the non-camouflage property subsumes the outer totalistic property. Furthermore, the transition rule of the CA N has only 12 clauses, which is almost half of the number of M. These results will make it easier to design DNA reactions corresponding to the transition rule.

The factor that can be credited for this result is the high expressiveness of the Boolean totalistic form. For example, the transition from cell state 2 to cell state Y is common in both of M and the CA in [2]. In the outer totalistic form, this transition is represented by 7 clauses. The number of states of neighboring cells changes depending on where the wire cell is, so a distinct clause corresponding to each situation is required. In our Boolean totalistic form, the transition is represented by just one clause, i.e., the fifth clause in Table 1.

We succeeded in constructing asynchronous CAs suitable for implementation by a reaction-diffusion system, but there is room for improvement. First, it may be necessary to further simplify the transition rule so that the number of reactions is reduced. Second, execution of distributed computation on an asynchronous non-camouflage CA has not been realized yet. Although CAs are regarded as models of distributed computation, M can only simulate deterministic Turing machines. Computation by N is more indirect and far from distributed computation. Finally, proving the Turing-completeness of a non-camouflage CA restricted to finite configurations is an interesting open problem.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank Yuichi Nishiwaki for useful discussions while constructing N. We also would like to thank the anonymous reviewers for their useful comments. This work was supported by a Grant-in-Aid for Scientific Research on Innovative Areas “Molecular Robotics” (No. 15H00825 and No. 24104005) and a Grant-in-Aid for Challenging Exploratory Research (No. 17K19961) of The Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] E. Goles, N. Ollinger, G. Theyssier, Introducing freezing cellular automata, in: Proceedings of 21st International Workshop on Cellular Automata and Discrete Complex Systems (AUTOMATA2015), 2015, pp. 65–73.
[2] T. Isokawa, F. Peper, I. Kawamata, N. Matsui, S. Murata, M. Hagiya, Universal totalistic asynchonous cellular automaton and its possible implementation by DNA, in: International Conference on Unconventional Computation and Natural Computation, Springer, 2016, pp. 182–195.
[3] I. Kawamata, T. Hosoya, F. Takabatake, K. Sugawara, S.-I. Nomura, T. Isokawa, F. Peper, M. Hagiya, S. Murata, Pattern formation and computation by autonomous chemical reaction diffusion model inspired by cellular automata, in: 2016 Fourth International Symposium on Computing and Networking, CANDAR, IEEE, 2016, pp. 215–221.
[4] M. Hagiya, S. Wang, I. Kawamata, S. Murata, T. Isokawa, F. Peper, K. Imai, On DNA-based gellular automata, in: International Conference on Unconventional Computation and Natural Computation, Springer, 2014, pp. 177–189.
[5] S. Wang, K. Imai, M. Hagiya, On the composition of signals in gellular automata, in: 2014 Second International Symposium on Computing and Networking, IEEE, 2014, pp. 499–502.
[6] L. Priese, Automata and concurrency, Theor. Comput. Sci. 25 (3) (1983) 221–265.
[7] N. Fates, A guided tour of asynchronous cellular automata, in: International Workshop on Cellular Automata and Discrete Complex Systems, Springer, 2013, pp. 15–30.
[8] A. Dennunzio, E. Formenti, L. Manzoni, G. Mauri, A.E. Porreca, Computational complexity of finite asynchronous cellular automata, Theor. Comput. Sci. 664 (2017) 131–143.
[9] A. Dennunzio, E. Formenti, L. Manzoni, Computing issues of asynchronous CA, Fundam. Inform. 120 (2) (2012) 165–180.
[10] A. Dennunzio, E. Formenti, L. Manzoni, G. Mauri, m-Asynchronous cellular automata: from fairness to quasi-fairness, Nat. Comput. 12 (4) (2013) 561–572.
[11] J. Kari, Theory of cellular automata: a survey, Theor. Comput. Sci. 334 (1–3) (2005) 3–33.
[12] A. Dennunzio, From one-dimensional to two-dimensional cellular automata, Fundam. Inform. 115 (1) (2012) 87–105.
[13] A. Dennunzio, E. Formenti, M. Weiss, Multidimensional cellular automata: closing property, quasi-expansivity, and (un) decidability issues, Theor. Comput. Sci. 516 (2014) 40–59.
[14] K. Morita, A simple universal logic element and cellular automata for reversible computing, in: International Conference on Machines, Computations, and Universality, Springer, 2001, pp. 102–113.
[15] B. Chopard, Cellular automata and lattice Boltzmann modeling of physical systems, in: Handbook of Natural Computing, 2012, pp. 287–331.
[16] G. Cattaneo, A. Dennunzio, F. Farina, A full cellular automaton to simulate predator-prey systems, in: International Conference on Cellular Automata, Springer, 2006, pp. 446–451.
[17] F. Farina, A. Dennunzio, A predator-prey cellular automaton with parasitic interactions and environmental effects, Fundam. Inform. 83 (4) (2008) 337–353.
[18] A. Bandyopadhyay, R. Pati, S. Sahu, F. Peper, D. Fujita, Massively parallel computing on an organic molecular layer, Nat. Phys. 6 (5) (2010) 369–375.
[19] S. Adachi, F. Peper, J. Lee, Universality of hexagonal asynchronous totalistic cellular automata, in: International Conference on Cellular Automata, Springer, 2004, pp. 91–100.
[20] M. Cook, Universality in elementary cellular automata, Complex Syst. 15 (1) (2004) 1–40.