Turing-Completeness of Asynchronous Non-camouflage Cellular Automata

Tatsuya Yamashita, Teijiro Isokawa, Ferdinand Peper, Ibuki Kawamata, Masami Hagiya

Turing-Completeness of Asynchronous Non-Camouflage Cellular Automata

Tatsuya Yamashita[1], Teijiro Isokawa[2], Ferdinand Peper[3], Ibuki Kawamata[4], and Masami Hagiya[1]

[1] University of Tokyo,
[2] University of Hyogo,
[3] NICT and Osaka University,
[4] Tohoku University,

Abstract. Asynchronous Boolean totalistic cellular automata have recently attracted attention as promising models for the implementation of 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 is stronger than 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. This indicates 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 transition rules 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 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 *Gellular Automata* [1, 6].

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 transition rules.

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 is stronger than the Boolean totalistic property. We present an asynchronous non-camouflage CA and prove that it is Turing-complete.

This paper is organized as follows. Section 2 gives the formal definitions of the used concepts. This is followed by a description of the proposed CA in Section 3, and the proof that it is Turing-universal. This paper finishes with a discussion in Section 4.

2 Preliminaries

2.1 Asynchronous CA

In this paper, we follow the terminology used in [5]. 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.

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 [5] 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

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

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 image 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 [5]. (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. A CA is a state-system whose states can be considered as an arrangement in a certain topology [5] of cell states, whereby the transition relation is determined by applying a local transition rule to each cell simultaneously. 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 $t_1, t_2 \in T$,

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

For an asynchronous CA A, S_A and f_A are called the *space of cell states* and the *transition rule*, respectively.

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

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 $s_j, \dots$. If there is no form 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 form whose left-hand side are 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 is stronger than the outer totalistic property.

As discussed in Section 1, the Boolean totalistic property is still not strong enough for our purposes. We define stronger property, non-camouflage.

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.

2.3 Priese System

In Section 3, we construct an asynchronous non-camouflage CA, and prove that it is Turing-complete. In [5], 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 7 (s-automaton). *A tuple $A = (I, O, S, \to)$ is called an s-automaton if*

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

An s-automaton $A = (I, O, S, \to)$ 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. $((x, s), (y, s')) \in \to$ is denoted as $(x, s) \to (y, s')$. 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 $A = (I, O, S, \to)$ can be considered as a state-system $((I \sqcup O) \times S, \to)$[6].

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

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

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

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

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 each other, we define two operations over s-automata, product and feed-back.

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

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

Definition 8 (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 9 (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 $A_1, \ldots, A_n$, he or she 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 10 (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.

Priese System is defined as the Normed Network over s-automata K and E. It is known that any finite-state s-automaton belongs to Priese System [5]. Next we define the infinite chain made of two s-automata A and B, where A and infinite copies of B are connected in sequence.

Definition 11 (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 also 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 [4].

3 Proposing asynchronous non-camouflage CA

3.1 Proposed CA M

In this subsection, we present an asynchronous non-camouflage CA.

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 state 0 to the 20 states appearing in the table of transition rules.

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 1. The transition rule of M.

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

Since a cell in the state 0 will never be changed to another state or influence transitions of neighboring cells, we assume that almost all cells are in the cell state 0 and such cells are not drawn in the figures.

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 made of two s-automata in Priese System with an initial configuration that is periodic except for a finite area. The Turing-completeness of M follows from this fact.

Wires and signals Cells with the state $1 \in S_{\mathrm{M}}$ extending linearly are called a wire. Figure 1 shows an image 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 (see Figure 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.

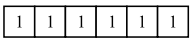

Fig. 1. A wire.

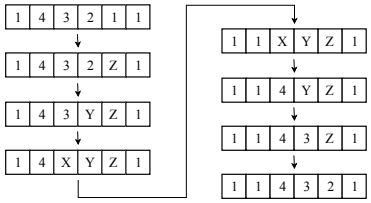

Fig. 2. A signal on a wire.

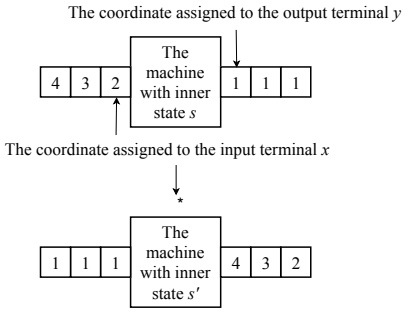

Fig. 3. An image of wire-based simulation.

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 represent such situations. Figure 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.

Definition 12 (wire-based simulation). *Let $A = (I_A, O_A, S_A, \to_A)$ be an s-automaton and $F : (I_A \sqcup O_A) \times S_A \to \wp(S_{\mathrm{M}}^{\mathbb{Z}\times\mathbb{Z}})$ be a simulation function of A by* M*. F is called a wire-based simulation function if there are functions $G : S_A \to S_{\mathrm{M}}^{\mathbb{Z}\times\mathbb{Z}}$, $g : I_A \sqcup O_A \to \mathbb{Z} \times \mathbb{Z}$ and integers $x_{min}, x_{max}, y_{min}, y_{max} \in \mathbb{Z}$ such that*

(i) for each state $s \in S_A$ of A, there is no transition from $G(s)$,
(ii) $\forall s \in S_A.\, \forall x, y \in \mathbb{Z}.\, G(s)(x, y) = 0 \vee (x_{min} \le x \le x_{max} \wedge y_{min} \le y \le y_{max})$,
(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* $io \in (I_A \backslash \{i\}) \sqcup O_A$*, 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 12. So we regard the pair of G and g as F. In figures, $G(s)$ for a state $s \in S_A$ is shown as an arrangement of cell states and $g(io)$ is shown by an arrow pointing at the cell corresponding to the input/output terminal io.

Simulation of K We construct a wire-based simulation of the s-automaton K now. Figure 4 shows the state $G_{\mathrm{K}}(0)$ and the coordinate $g_{\mathrm{K}}(io)$ for each terminals $io \in I_{\mathrm{K}} \sqcup O_{\mathrm{K}}$. They give wire-based simulation F_{K} of the s-automaton K in state $0 \in S_{\mathrm{K}}$. Figure 5 shows the state $F_{\mathrm{K}}(0,0)$. The states simulating K with a signal in the other input/output terminals are also constructed in the same way.

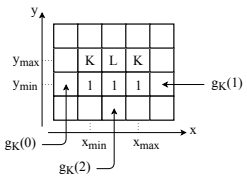

Fig. 4. $G_{\mathrm{K}}(0)$ and g_{K}.

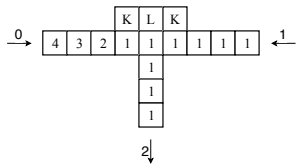

Fig. 5. $F_{\mathrm{K}}(0,0)$.

Now we confirm that the function F_{K} determined by Figure 4 and the conditions of Definition 12 is a wire-based simulation function of K. Since there is no rule in Table 1 that is applicable to a cell in $G_{\mathrm{K}}(0)$, the state $G_{\mathrm{K}}(0)$ cannot be changed. Thus, the condition (i) of Definition 12 is satisfied. Recall that the cells which are not drawn in figures are assumed to be in state $0 \in S_{\mathrm{M}}$. That means the condition (ii) of Definition 12 is satisfied if the frame of F_{K} is determined by $x_{min}, x_{max}, y_{min}$ and y_{max} in Figure 4. The condition (iii) of Definition 12 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.

Figure 6 shows the transitions that can be made if the initial state is $F_{\mathrm{K}}(0,0)$. Many transitions are presented by $\rightarrow^*$ to save space. These omitted transitions are almost the same as the transitions shown in Figure 2. Any state t' which

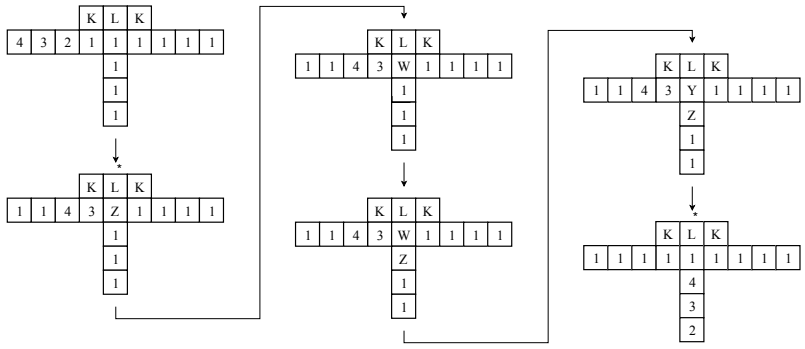

Fig. 6. The transitions from $F_{\mathrm{K}}(0,0)$ to $F_{\mathrm{K}}(2,0)$.

satisfies $F_{\mathrm{K}}(0,0) \rightsquigarrow_{F_{\mathrm{K}}} t'$ appears in the transitions shown in Figure 6, and any state t' in this 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.

Simulation of E The s-automaton E with the inner state u or d is simulated by the arrangement of cell states shown in Figure 7 or Figure 8, respectively. The unique difference of 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 Figure 7 and Figure 8 is a wire-based simulation of E.

The procedures to confirm that transitions starting from a state on M satisfy 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 checked the conditions of Definition 2.

Crossing Since we are dealing with two-dimensional space, it may be difficult to connect two terminals with a wire. We solve this problem by constructing a crossing, which allows two wires to cross each other. The construction of a crossing is shown in Figure 9.

Figure 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 rule 1 of Table 1. That is the reason why a signal progresses straight at the center of a crossing. These transitions are non-deterministic, but any succeeding state transfers to the state having a signal in the opposite side of the initial state. Thanks to crossings, we can connect terminals freely.

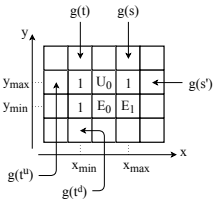

Fig. 7. $G_{\mathrm{E}}(u)$.

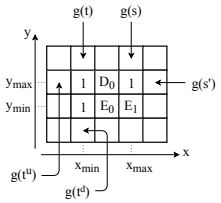

Fig. 8. $G_{\mathrm{E}}(d)$.

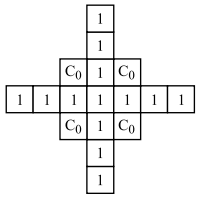

Fig. 9. Crossing.

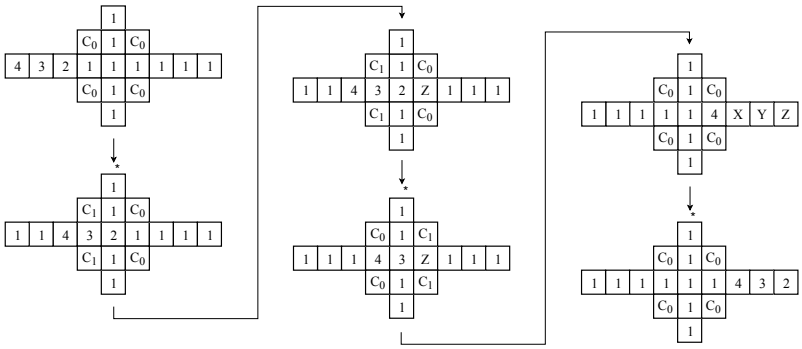

Fig. 10. A signal on a crossing.

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 $A = (I_A, O_A, S_A, \to_A)$ and $B = (I_B, O_B, S_B, \to_B)$ 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 figure of Figure 11 shows the image of $G_{A\otimes B}(s, s')$.

Let $A = (I_A, O_A, S_A, \to_A)$, $x \in I_A$ and $y \in O_A$ be an s-automaton, its input terminal and its output terminal, respectively. 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 as follows. First, a wire is extended from $g_A(y)$ to $g_A(x)$. Next, the frame of $F_{A^x_y}$ 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

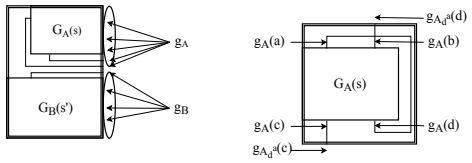

Fig. 11. Left and right figures show $G_{A\otimes B}(s, s')$ and $G_{A_d^a}(s)$, respectively. Their frames are represented by rectangles drawn with double lines.

coordinate $g_{A_y^x}(io)$ is determined by the end of the extended wire. If necessary, wires cross each other by using a crossing.

The right figure of Figure 11 shows an image 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 the crossing must be used at the point.

Infinite chain Let A and B be s-automata in Priese System. There exist wire-based simulation functions F_A and F_B. Assume that A and B satisfy the assumptions of Definition 11. A wire-based simulation of the infinite chain made of A and B can be constructed by putting G_A and infinite copies of G_B in sequence and connecting the corresponding terminals to each other.

4 Discussion

This paper has presented an asynchronous non-camouflage CA, which is suitable for implementation by a reaction-diffusion system. The automaton has been obtained by making some changes to the asynchronous totalistic CA presented in [2]. The asynchronous CA presented in this paper is Turing-complete as expected.

The transition function of the asynchronous CA proposed in this paper is represented by less rules than the previous one is; 23 rules of the transition function of the former CA are less than half of 57 rules of that of the latter CA. This is a surprising result because the non-camouflage property is stronger than the outer totalistic property. This result will make it easy to design DNA reactions corresponding to the transition function.

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 asynchronous CA. In outer totalistic form, this transition is represented by 7 rules. The number of states of neighboring cells changes depending on where the wire cell is, so a distinct rule corresponding to each situation is required. In our Boolean totalistic form, the transition is represented by just one rule (see rule 5 in Table 1).

We succeeded in constructing an asynchronous CA suitable for implementation by a reaction-diffusion system, but several practical obstacles remain until

this implementation can be realized. The proposed cellular automaton has 21 cell states, but it is necessary to reduce the number of cell states to actually be able to implement the CA by a reaction-diffusion system with DNA molecules. It is also necessary to further simplify the transition rule so that the number of reactions is reduced. Finally, the intrinsic universality of non-camouflage CA, i.e., its Turing-completeness restricted to finite configurations is an interesting open problem.

Acknowledgements This work was supported by a Grant-in-Aid for Scientific Research on Innovative Areas "Molecular Robotics" (No. 15H00825 and No. 24104005) of The Ministry of Education, Culture, Sports, Science, and Technology, Japan.

References

1. M. Hagiya, S. Wang, I. Kawamata, S. Murata, T. Isokawa, F. Peper, and K. Imai. On DNA-based gellular automata. In *International Conference on Unconventional Computation and Natural Computation*, pages 177–189. Springer, 2014.
2. T. Isokawa, F. Peper, I. Kawamata, N. Matsui, S. Murata, and M. Hagiya. Universal totalistic asynchonous cellular automaton and its possible implementation by DNA. In *15th International Conference on Unconventional Computation and Natural Computation: (UCNC 2016)*, pages 182–195, 2016.
3. I. Kawamata, T. Hosoya, F. Takabatake, K. Sugawara, S. Nomura, T. Isokawa, F. Peper, M. Hagiya, and S. Murata. Pattern formation and computation by autonomous chemical reaction diffusion model inspired by cellular automata. In *Computing and Networking (CANDAR), 2016 Fourth International Symposium on*, pages 215–221. IEEE, 2016.
4. Kenichi Morita. A simple universal logic element and cellular automata for reversible computing. In *International Conference on Machines, Computations, and Universality*, pages 102–113. Springer, 2001.
5. L. Priese. Automata and concurrency. *Theoretical Computer Science*, 25(3):221 – 265, 1983.
6. S. Wang, K. Imai, and M. Hagiya. On the composition of signals in gellular automata. In *Computing and Networking (CANDAR), 2014 Second International Symposium on*, pages 499–502. IEEE, 2014.