How Can We Construct Reversible Turing Machines in a Very Simple Reversible Cellular Automaton?

Kenichi Morita(✉)

Hiroshima University, Higashi-Hiroshima 739-8527, Japan
km@hiroshima-u.ac.jp

Abstract. A reversible cellular automaton (RCA) is an abstract spatiotemporal model of a reversible world. Using the framework of an RCA, we study the problem of how we can elegantly compose reversible computers from simple reversible microscopic operations. The CA model used here is an *elementary triangular partitioned CA* (ETPCA), whose spatial configurations evolve according to an extremely simple local transition function. We focus on the particular reversible ETPCA No. 0347, where 0347 is an ID number in the class of 256 ETPCAs. Based on our past studies, we explain that reversible Turing machines (RTMs) can be constructed in a systematic and hierarchical manner in this cellular space. Though ETPCA 0347 is an artificial CA model, this method gives a new vista to find a pathway from a reversible microscopic law to reversible computers. In particular, we shall see that RTMs can be easily realized in a unique method by using a *reversible logic element with memory* (RLEM) in the intermediate step of the pathway.

Keywords: Reversible cellular automaton · Elementary triangular partitioned cellular automaton · Reversible Turing machine · Reversible logic element with memory

1 Introduction

We investigate the problem of composing reversible Turing machines (RTMs), a model of reversible computers, from a simple reversible microscopic law. In particular, we study how simple the reversible microscopic law can be, and how elegantly RTMs are designed in a given environment. For this purpose, we use a reversible cellular automaton (RCA) as an abstract discrete model of a reversible world. Here, we consider a special type of an RCA called a reversible elementary triangular partitioned cellular automaton (ETPCA) having only four simple local transition rules. Thus, in this framework, the problem becomes how to construct RTMs using only such local transition rules.

However, since the local transition rules are so simple, it is difficult to directly design RTMs in this cellular space. One method of solving this problem is to divide a pathway, which starts from a reversible microscopic law (i.e., local transition rules) and leads to reversible computers (i.e., RTMs), into several segments. Namely, we put several suitable conceptual levels on the pathway, and solve a subproblem in each level as in Fig. 1. Five levels are supposed here.

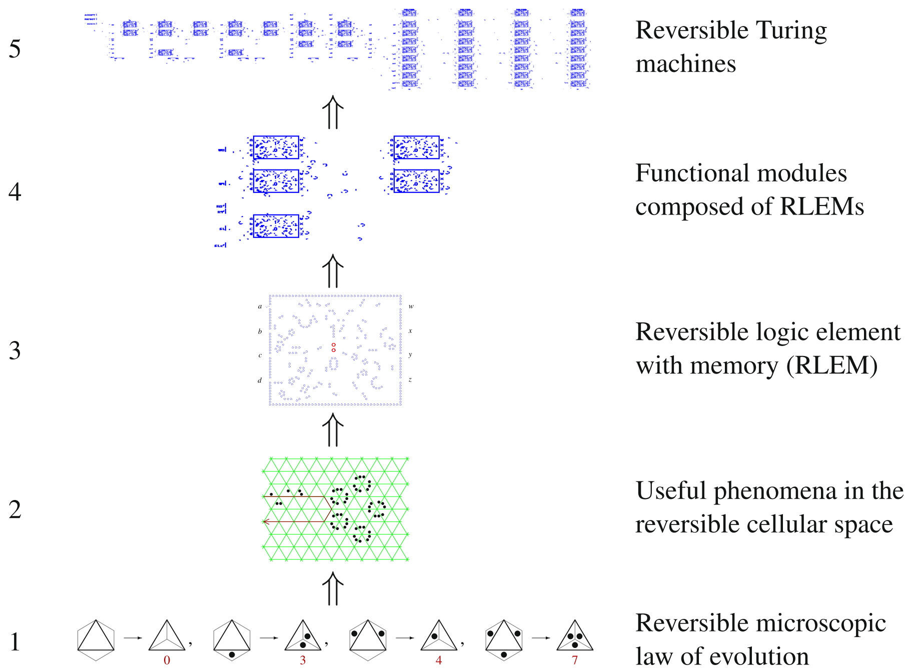
5
Reversible Turing machines
4
Functional modules composed of RLEMs
3
Reversible logic element with memory (RLEM)
2
Useful phenomena in the reversible cellular space
1
Reversible microscopic law of evolution
0
3
4
7

Fig. 1. A pathway from a reversible microscopic law to reversible computers

- Level 1: A reversible ETPCA, a model of a reversible world, is defined. Its local transition rules are considered as a microscopic law of evolution.
- Level 2: Various experiments in this cellular space are done to see how configurations evolve. By this, useful patterns and phenomena are found.
- Level 3: The phenomena found in the level 2 are used as gadgets to compose a logical primitive. Here, we make a reversible logic element with memory (RLEM), rather than a reversible logic gate, combining these gadgets.
- Level 4: Functional modules for RTMs are composed out of RLEMs. These modules are constructed easily and elegantly by using RLEMs.
- Level 5: RTMs are systematically built by assembling the functional modules created in the level 4, and then realized in the reversible cellular space.

In this way, we can construct RTMs from very simple local transition rules in a systematic and modularized method. Though ETPCA 0347 is an artificial CA model, it will give an insight to find a pathway even in a different situation.

In particular, we can see that it is important to give intermediate conceptual levels appropriately on the pathway to design reversible computers elegantly.

2 Reversible Cellular Automaton

In this section, using the framework of an RCA, a reversible microscopic law of evolution is given, which corresponds to the level 1 in Fig. 1. After a brief introduction to a CA and an RCA, we define a specific reversible ETPCA with the ID number 0347. Note that formal definitions on CA, ETPCA, and their reversibility are omitted here. See, e.g., [8] for their precise definitions.

2.1 Cellular Automaton and Its Reversibility

A *cellular automaton* (CA) is an abstract discrete model of spatiotemporal phenomena. It consists of an infinite number of identical finite automata called *cells* placed uniformly in the space. Each cell changes its state depending on the states of its neighbor cells using a *local function*, which is described by a set of local transition rules. Applying the local function to all the cells simultaneously, a *global function* that specifies the transition among *configurations* (i.e., whole states of the infinite cellular space) is obtained. Figure 2 shows a two-dimensional CA whose cells are square ones. In this figure each cell changes its state depending on the states of its five neighbor cells (including itself).

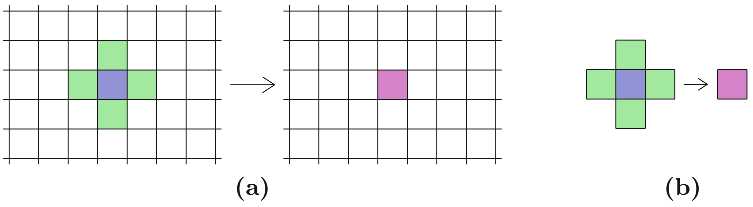

Fig. 2. Two-dimensional cellular automaton with square cells. **(a)** Its cellular space, and **(b)** its local transition rule

A *reversible cellular automaton* (RCA) is a CA whose global function is injective. Hence, there is no pair of distinct configurations that go to the same configuration by the global function. However, it is generally difficult to design an RCA if we use the standard framework of CAs. In particular, it is known that the problem whether the global function of a given two-dimensional CA is injective is undecidable [4]. In Sect. 2.2 we use a *partitioned CA* (PCA) [11], which is a subclass of the standard CA, for designing an RCA.

2.2 Triangular Partitioned Cellular Automaton (TPCA)

Hereafter, we use a *triangular partitioned cellular automaton* (TPCA) [3,9]. In a TPCA, a cell is an equilateral triangle, and is divided into three parts, each of which has its own state set. The next state of a cell is determined by the present states of the three adjacent parts of the neighbor cells (not by the whole states of the three adjacent cells) as shown in Fig. 3.

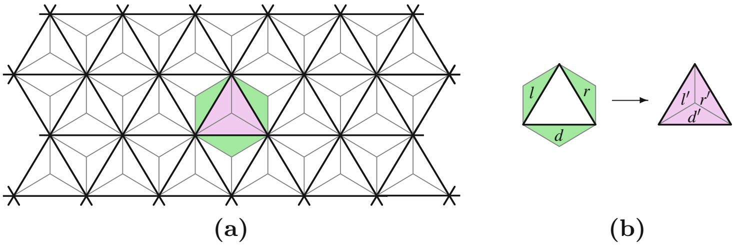

Fig. 3. Triangular partitioned cellular automaton. **(a)** Its cellular space, and **(b)** its local transition rule

A *local function* of a TPCA is defined by a set of local transition rules of the form shown in Fig. 3**(b)**. Applying the local function to all the cells the *global function* of the TPCA is obtained.

The reason why we use a TPCA is as follows. First, since the number of edge-adjacent cells of each cell is only three, its local function can be much simpler than that of a CA with square cells. Second, the framework of PCA makes it feasible to design a reversible CA, since it is easy to show Lemma 1. Hence, TPCA is suited for studying the problem how simple a computationally universal RCA can be.

Lemma 1. *Let P be a PCA. Let f and F be its local and global functions, respectively. Then, F is injective if and only if f is injective.*

Note that this lemma was first shown in [11] for a one-dimensional PCA. The lemma for TPCA was given in [8].

2.3 Elementary Triangular Partitioned Cellular Automaton (ETPCA), in Particular, ETPCA 0347

An *elementary triangular partitioned cellular automaton* (ETPCA) is a subclass of a TPCA such that each part of a cell has the state set $\{0, 1\}$, and its local function is *isotropic* (i.e., *rotation-symmetric*) [3,9].

Figure 4 shows the four local transition rules of an ETPCA with the ID number 0347, which is denoted by ETPCA 0347. Note that since an ETPCA is isotropic, local transition rules obtained by rotating both sides of the rules by a multiple of 60° are omitted here. Therefore, the local function is completely

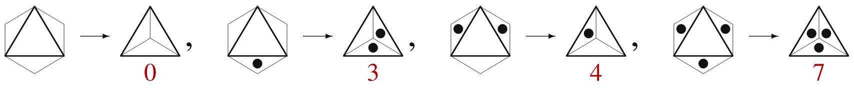

Fig. 4. Local function of ETPCA 0347 defined by the four local transition rules [9]. States 0 and 1 are indicated by a blank and ●

determined by these four rules. Hereafter, we use ETPCA 0347 as our model of a reversible world.

Generally, each ETPCA has an ID number of the form $wxyz$ ($w, z \in \{0, 7\}$, $x, y \in \{0, 1, ..., 7\}$), and is denoted by ETPCA $wxyz$. Figure 5 shows how an ID number corresponds to the four local transition rules. Note that w and z must be 0 or 7, since an ETPCA is isotropic. Thus, there are 256 ETPCAs in total.

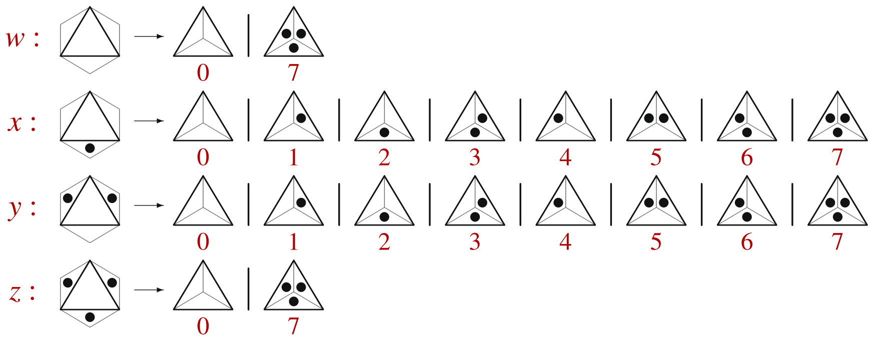

Fig. 5. Correspondence between the ID number $wxyz$ and the four local transition rules of ETPCA $wxyz$ ($w, z \in \{0, 7\}$, $x, y \in \{0, 1, ..., 7\}$). Vertical bars indicate alternatives of a right-hand side of each local transition rule

We can verify that the local function of ETPCA 0347 is injective, since there is no pair of local transition rules that have the same right-hand side (see Fig. 4). Therefore, its global function is injective, and thus ETPCA 0347 is *reversible*. We can see every configuration of ETPCA 0347 has exactly one predecessor.

3 Useful Patterns and Phenomena in the Reversible Cellular Space of ETPCA 0347

Here, we make various experiments in the cellular space of ETPCA 0347, and look for useful patterns and phenomena. It corresponds to the level 2 in Fig. 1. Note that most experiments described below were firstly done in [6,9].

3.1 Useful Patterns in ETPCA 0347

A *pattern* is a finite segment of a configuration. It is known that there are three kinds of patterns in ETPCA 0347 since it is reversible [9]. They are a periodic pattern, a space-moving pattern, and an (eventually) expanding pattern. Here, a periodic pattern and a space-moving pattern are important. As we shall see below two periodic patterns called a *block* and a *fin*, and one space-moving pattern called a *glider* are particularly useful.

A *periodic pattern* is one such that the same pattern appears at the same position after p steps of time ($p > 0$). The minimum of such p's is the *period* of the pattern. The pattern that appears at time t $(0 \leq t \leq p-1)$ is called the pattern of *phase* t. A periodic pattern of period 1 is called a *stable pattern*.

A *block* is a stable pattern shown in Fig. 6. We can verify that a block does not change its pattern by the application of the local function given in Fig. 4.

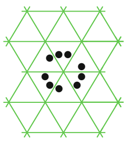

Fig. 6. A stable pattern called a *block* in ETPCA 0347 [9]

A *fin* is a periodic pattern of period 6 given in Fig. 7. We can also verify that a fin changes its pattern as shown in this figure.

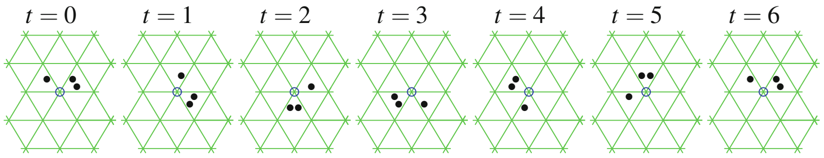

Fig. 7. A periodic pattern of period 6 called a *fin* in ETPCA 0347 [9]. It rotates around the point indicated by ◦. The pattern at t $(0 \leq t \leq 5)$ is called a fin of phase t

A *space-moving pattern* is one such that the same pattern appears at a different position after p steps of time. The minimum of such p's is the period. The pattern that appears at time t $(0 \leq t \leq p-1)$ is called the pattern of *phase* t.

A *glider* is a space-moving pattern shown in Fig. 8. It is the most useful pattern. If we start with the pattern given at $t = 0$ in Fig. 8, the same pattern appears at $t = 6$, and its position is shifted rightward. Thus, it swims like a fish or an eel in the reversible cellular space. It will be used as a *signal* when we implement a logic element in ETPCA 0347.

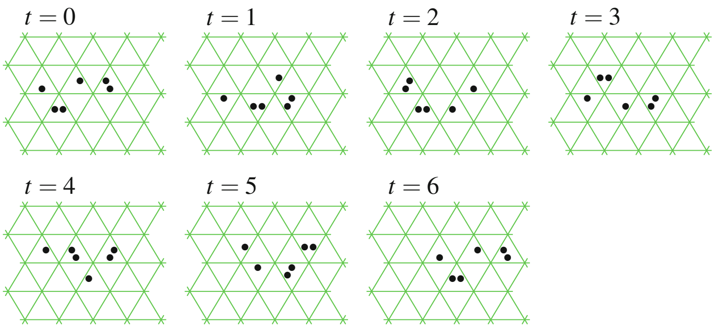

Fig. 8. A space-moving pattern of period 6 called a *glider* in ETPCA 0347 [9]. It moves rightward. The pattern at t ($0 \leq t \leq 5$) is called a glider of phase t

3.2 Interacting Patterns in ETPCA 0347 to Find Useful Phenomena

Next, we make various experiments of interacting a glider with blocks or a fin. By this, we can find several phenomena that can be used as "gadgets" for composing a logic element in Sect. 4.4.

It should be noted that despite the simplicity of the local function (Fig. 4) time evolutions of configurations in ETPCA 0347 are generally very complex. Therefore, it is very hard to follow evolution processes by paper and pencil. We developed an emulator for ETPCA 0347 that works on a general-purpose high-speed CA simulator *Golly* [15]. The emulator file and many pattern files are available in [7] (though its new version has not yet been uploaded).

First, we create several gadgets for controlling the move direction of a glider. Figure 9 shows a 60°-*right-turn gadget* composed of two blocks. It is newly introduced in this paper. Using several (rotated and unrotated) copies of this gadget, the move direction of a glider is changed freely.

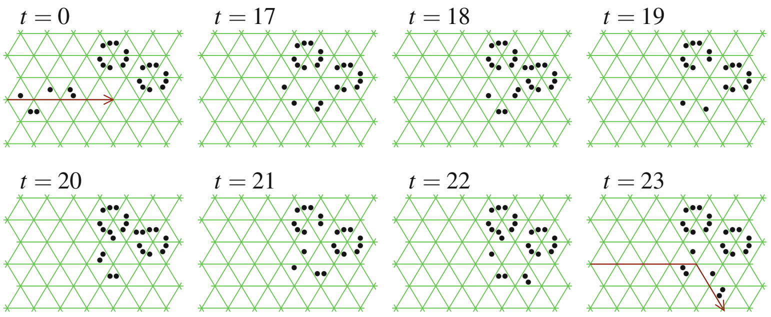

Fig. 9. 60°-right-turn gadget for a glider composed of two blocks

We can make a 120°-right-turn gadget as in Fig. 10. If we collide a glider with a sequence of two blocks as shown in Fig. 10 ($t = 0$), it is first decomposed into a "body" (left) and a fin (right) ($t = 56$). The body rotates around the point indicated by a small circle, and the fin goes around the blocks clockwise. Finally, the body meets the fin, and a glider is reconstructed. The resulting glider moves to the south-west direction ($t = 334$). A similar 120°-right-turn gadget is also possible by using a sequence of three blocks or five blocks.

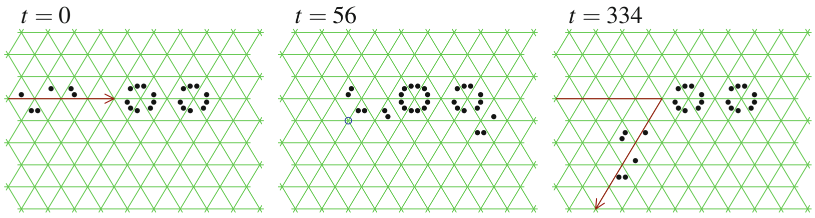

Fig. 10. 120°-right-turn gadget for a glider composed of two blocks [9]

It should be noted that 120°-right-turn is also realized by using two (rotated and unrotated) copies of a 60°-right-turn gadget. Furthermore, in the latter implementation, right-turn is performed more quickly. Hence a 60°-right-turn gadget is more useful in this respect. However, a 120°-right-turn gadget can be used as an interface between a bidirectional signal path and unidirectional signal paths as shown in Fig. 11. Such a gadget is necessary for constructing a logic element in Sect. 4.4, since a fin-shifting (Fig. 14) uses a bidirectional path.

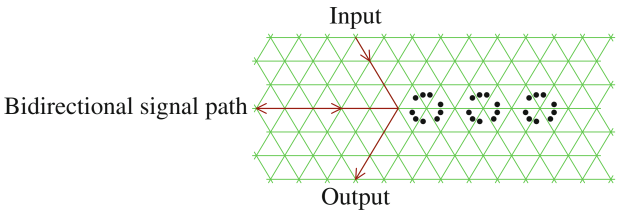

Fig. 11. Interface gadget between bidirectional and unidirectional signal paths [9]

Figures 12 and 13 show a *backward-turn gadget* and a *U-turn gadget*. A 120°-*left-turn gadget* is also given in [9], but it is omitted here. These gadgets are needed when we want to shift the phase of a glider (see [9]).

Next, we make experiments of interacting a glider with a fin. Figure 14 shows that shifting the position of a fin is possible if we collide a glider with the right phase to it [6]. Thus, a fin can be used as a kind of memory, where memory states (say 0 and 1) are distinguished by the positions of the fin. In this case, change of the memory state can be done by shifting the fin appropriately.

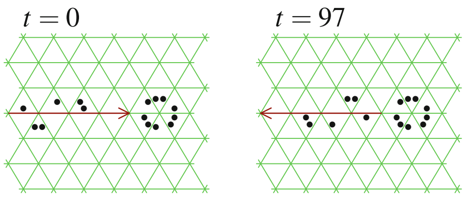

Fig. 12. Backward-turn gadget for a glider [9]

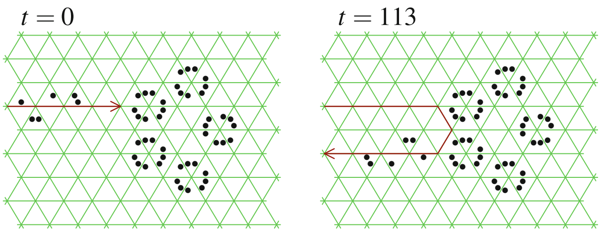

Fig. 13. U-turn gadget for a glider [9]

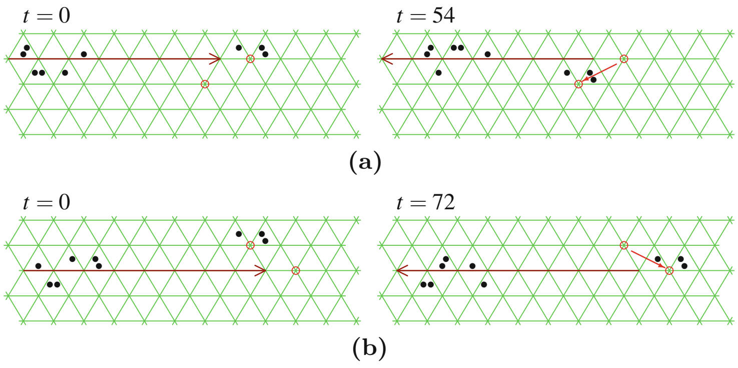

Fig. 14. Shifting a fin by a glider [6]. **(a)** Pulling, and **(b)** pushing

4 Composing Reversible Logic Element with Memory

Using useful phenomena found in Sect. 3, a logic element is composed. It corresponds to the level 3 in Fig. 1. Here, a reversible logic element with memory, rather than a reversible logic gate, is implemented in ETPCA 0347.

4.1 Reversible Logic Element with Memory (RLEM)

A *reversible logic element with memory* (RLEM) [8,12] is a kind of a reversible finite automaton having both input and output ports, which is sometimes called a reversible sequential machine of Mealy type. A *sequential machine* M is defined by $M = (Q, \Sigma, \Gamma, \delta)$, where Q is a finite set of states, Σ and Γ are finite sets of input and output symbols, and $\delta : Q \times \Sigma \rightarrow Q \times \Gamma$ is a move function (Fig. 15**(a)**). If δ is injective, it is called a *reversible sequential machine* (RSM).

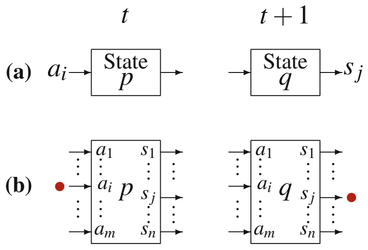

Fig. 15. (a) A sequential machine, and **(b)** an interpretation of it as a module having decoded input ports and output ports for utilizing it as a circuit module

To use an SM as a circuit module, we interpret it as the one with decoded input/output ports (Fig. 15**(b)**). Namely, for each input symbol, there exists a unique input port to which a signal (or a particle) is given. It is also the case for the output symbols. Therefore, we assume signals should not be given to two or more input ports at the same time. Its operation is undefined if such a case occurs. Of course, it could be extended so that an SM can receive two or more signals simultaneously. However, we do not do so, because we want to keep its operation as simple as possible. Moreover, as described in Sect. 4.3, RSMs are sufficiently powerful without such an extension.

An RLEM is an RSM that satisfies $|\Sigma| = |\Gamma|$. A 2-state RLEM (i.e., $|Q| = 2$) is particularly important, since it is simple yet powerful (see Sect. 4.3). In the following we shall use a specific 2-state 4-symbol RLEM No. 4-31.

Advantages of RLEMs over reversible logic gates are as follows. As described above, an RLEM receives only one signal at the same time, while a logic gate generally receives two or more signals simultaneously. Therefore, in the case of a logic gate, some synchronization mechanism is necessary so that signals arrive at exactly the same time at each gate. On the other hand, in the case of an

RLEM, a signal can arrive at any time, since a signal interacts with a state of the RLEM, not with another signal. An RLEM is thus completely different from a logic gate in this respect. By this, realization of an RLEM and its circuit becomes much simpler than that of a logic gate at least in an RCA. This problem will be discussed again in Sect. 4.5.

4.2 RLEM 4-31

Here, we consider a specific 2-state 4-symbol RLEM 4-31 such that $Q = \{0, 1\}$, $\Sigma = \{a, b, c, d\}$, $\Gamma = \{w, x, y, z\}$, and δ is defined as follows:

$$\delta(0, a) = (0, w),\ \delta(0, b) = (0, x),\ \delta(0, c) = (0, y),\ \delta(0, d) = (1, w),$$
$$\delta(1, a) = (1, x),\ \delta(1, b) = (0, z),\ \delta(1, c) = (1, z),\ \delta(1, d) = (1, y).$$

In RLEM 4-31, "4" stands for "4-symbol," and "31" is its serial number in the class of 2-state 4-symbol RLEMs [8].

The move function δ of RLEM 4-31 is represented in a graphical form as shown in Fig. 16**(a)**. Two rectangles in the figure correspond to the two states 0 and 1. Solid and dotted lines show the input-output relation in each state. If an input signal goes through a dotted line, then the state does not change (Fig. 16**(b)**). On the other hand, if a signal goes through a solid line, then the state changes (Fig. 16**(c)**).

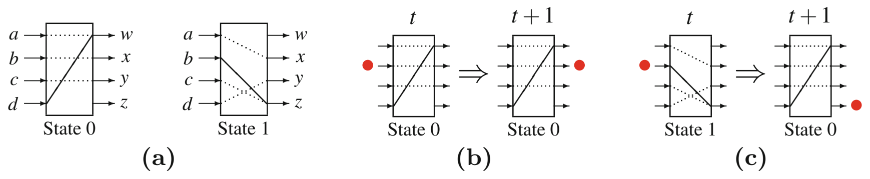

Fig. 16. RLEM 4-31, and its operation examples. **(a)** Two states of RLEM 4-31. **(b)** The case that the state does not change, and **(c)** the case that the state changes

4.3 Universality of RLEMs

There are infinitely many 2-state RLEMs if we do not restrict the number of input/output symbols. Hence, it is an important problem which RLEMs are sufficiently powerful, and which are not. Remarkably, it is known almost all 2-state RLEMs are universal.

An RLEM E is called *universal* if any RSM is composed only of E. The following results on the universality are known. First, every non-degenerate 2-state k-symbol RLEM is universal if $k > 2$ [12]. Second, among four non-degenerate 2-state 2-symbol RLEMs, three RLEMs 2-2, 2-3 and 2-4 have been proved to be non-universal [14] (but the set {2-3, 2-4} is universal [5]). Figure 17 summarizes these results. Note that the definition of *degeneracy* is given in [8,12].

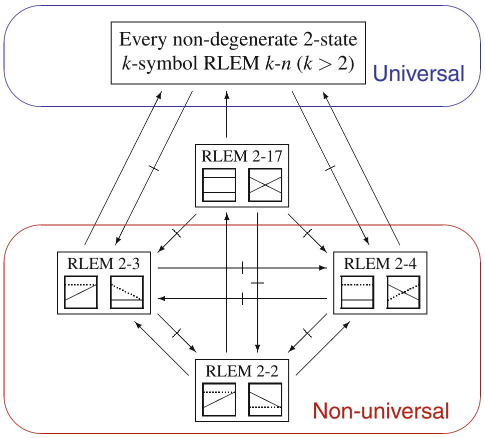

Fig. 17. Universality/non-universality of 2-state RLEMs [8,14]. Here, $A \to B$ ($A \not\to B$, respectively) indicates that A can (cannot) be simulated by B

4.4 Composing RLEM 4-31 in the Reversible ETPCA 0347

As we shall see in Sect. 5.2, a finite state control, and a memory cell, which corresponds to one tape square, of a reversible Turing machine (RTM) can be formalized as RSMs. Hence, any universal RLEM can be used to compose RTMs.

Here, we choose RLEM 4-31 for this purpose. The reason is as follows. First, it is known that a finite state control and a memory cell can be composed of RLEM 4-31 compactly [13]. Second, in RLEM 4-31, the number of transitions from one state to another (i.e., the number of solid lines in a graphical representation like Fig. 16) is only one in each state. By this, it becomes feasible to simulate RLEM 4-31 in ETPCA 0347 using the phenomena found in Sect. 3.

Figure 18 shows the complete pattern realized in ETPCA 0347 that simulates RLEM 4-31. Two circles in the middle of the pattern show possible positions of a fin. In this figure, the fin is at the lower position, which indicates the state of the RLEM 4-31 is 0. Many blocks are placed to form various turn gadgets shown in Figs. 9, 10, 11, 12 and 13. They are for controlling the direction and the phase of a glider.

In Fig. 18, a glider is given to the input port d. The path from d to the output port w shows the trajectory of the glider. The glider first goes to the north-east position. From there the glider moves to the south-west direction, and collides with the fin. By this, the fin is pulled upward by the operation shown in Fig. 14**(a)**. Then the glider goes to the south-east position. From there, it pushes the fin (Fig. 14**(b)**). By this, the fin moves to the position of the upper circle, which means the state changes from 0 to 1. The glider finally goes out from the port w. The whole process above simulates the one step move $\delta(0, d) = (1, w)$ of RLEM 4-31. Other cases are also simulated similarly. Note that the pattern in Fig. 18 is an improved version of the one given in [6] using the 60°-right-turn gadget in Fig. 9.

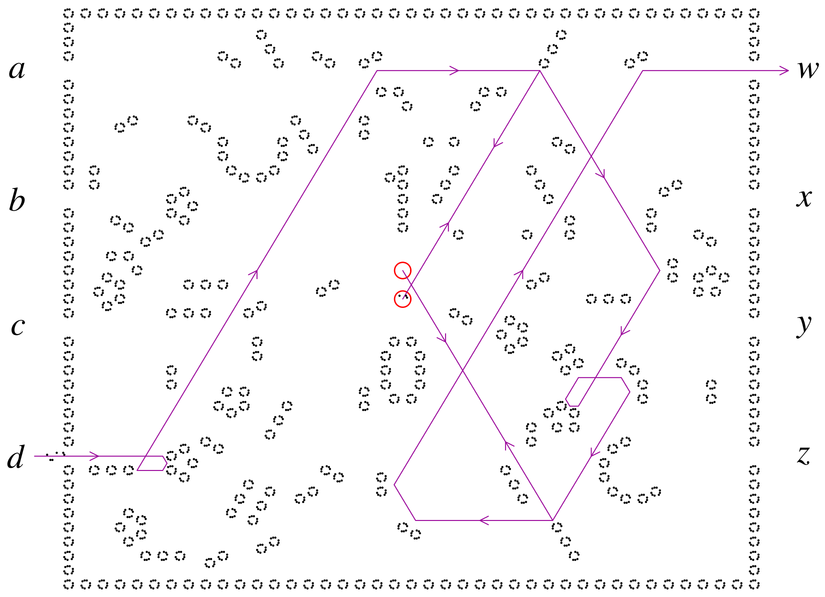

Fig. 18. RLEM 4-31 module implemented in ETPCA 0347

4.5 Comparing with the Method that Uses Reversible Logic Gates

In [9] it is shown that a Fredkin gate [2], a universal reversible logic gate, can be implemented in ETPCA 0347. There, a Fredkin gate is constructed out of two switch gates and two inverse switch gates. Therefore, it is in principle possible to implement a reversible Turing machine in ETPCA 0347 using a Fredkin gate. However, as stated in Sect. 4.1, it will become very complex since adjustment of signal timings at each gate is necessary. Here, we realize RLEM 4-31 in ETPCA 0347 using reversible logic gates, and compare it with the "direct implementation" given in Fig. 18.

Figure 19 shows two implementations of RLEM 4-31 simulated on Golly. The upper pattern is the direct implementation of RLEM 4-31 (the same as in Fig. 18), while the lower is constructed using four switch gates, two Fredkin gates, and four inverse switch gates. The size and the period of the former pattern are 82×163 and 6, and those of the latter are 462×1505 and 24216. Note that the period of the former is 6, since it contains a fin to memorize the state of RLEM 4-31. In the latter pattern, we need a circulating signal in the module to memorize the state since it is composed only of logic gates that are memoryless elements. This circulating signal determines the period 24216. Therefore, an input signal must be given at time t that satisfies $(t \bmod 24216) = 0$.

These values, of course, give only upper-bounds of the complexities of these two implementations. Hence they cannot be a proof for showing the advantage of the direct implementation. However, empirically, they suggest that it is much better *not* to use reversible logic gates in an RCA to make RTMs. Actually, if we compose a whole RTM using only reversible logic gates without introducing the notion of an RLEM, the resulting pattern will become quite complex.

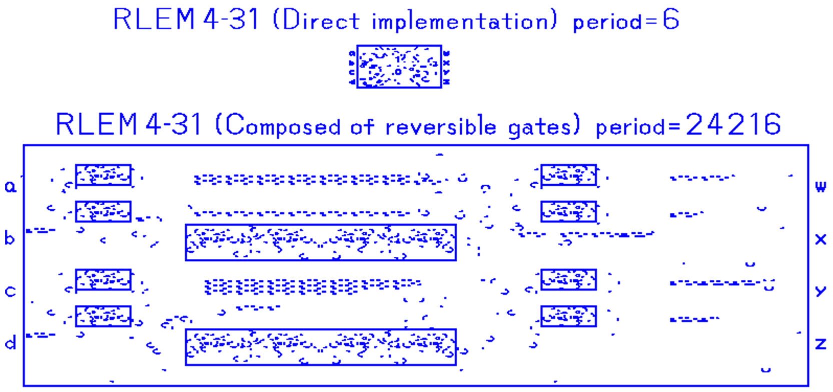
RLEM 4-31 (Direct implementation) period=6
RLEM 4-31 (Composed of reversible gates) period=24216
a
b
c
d
w
x
y
z

Fig. 19. Comparing the direct implementation of RLEM 4-31 (the upper pattern) with the one using reversible logic gates (the lower pattern) simulated on Golly

5 Making Reversible Turing Machines in ETPCA 0347

Here we realize reversible Turing machines in the cellular space of ETPCA 0347. To do it systematically, functional modules for RTMs are composed out of RLEM 4-31, and then they are assembled to make circuits that simulate RTMs. Finally the circuits are embedded in the space of ETPCA 0347. These steps correspond to the levels 4 and 5 in Fig. 1.

5.1 Reversible Turing Machine

A one-tape *Turing machine* (TM) in the quintuple form is defined by a system $T = (Q, S, q_0, F, s_0, \delta)$, where Q is a non-empty finite set of states, S is a non-empty finite set of tape symbols, q_0 is an initial state ($q_0 \in Q$), F is a set of final states ($F \subseteq Q$), and s_0 is a blank symbol ($s_0 \in S$). The item δ is a *move relation*, which is a subset of $(Q \times S \times S \times \{L, R\} \times Q)$, where L and R stand for left-shift and right-shift of the head. An element of δ is a *quintuple* of the form $[p, s, t, d, q]$. It means that if T is in the state p and reads the symbol s, then it writes the symbol t, shifts the head to the direction d and goes to the state q.

A TM T is called *deterministic*, if the following holds for any pair of distinct quintuples $[p_1, s_1, t_1, d_1, q_1]$ and $[p_2, s_2, t_2, d_2, q_2]$ in δ.

$$(p_1 = p_2) \Rightarrow (s_1 \neq s_2)$$

A TM T is called *reversible*, if the following holds for any pair of distinct quintuples $[p_1, s_1, t_1, d_1, q_1]$ and $[p_2, s_2, t_2, d_2, q_2]$ in δ.

$$(q_1 = q_2) \Rightarrow (d_1 = d_2 \wedge t_1 \neq t_2)$$

In the following we consider only deterministic TMs, and thus the word "deterministic" will be omitted. A TM that is reversible and deterministic is now called a *reversible Turing machine* (RTM). In an RTM, every computational configuration of it has at most one predecessor. See, e.g., [8] for a more detailed definition on RTMs.

Example 1. Consider the RTM $T_{\text{parity}} = (\{q_0, q_1, q_2, q_{\text{acc}}, q_{\text{rej}}\}, \{0, 1\}, q_0, \{q_{\text{acc}}\}, 0, \delta_{\text{parity}})$. The set δ_{parity} consists of the following five quintuples.

$$[q_0, 0, 1, R, q_1], [q_1, 0, 1, L, q_{\text{acc}}], [q_1, 1, 0, R, q_2], [q_2, 0, 1, L, q_{\text{rej}}], [q_2, 1, 0, R, q_1]$$

It is a very simple example of an RTM. In Sect. 5.3 a configuration that simulates it in ETPCA 0347 will be shown. It is easy to see that T_{parity} is deterministic and reversible. Assume a symbol string $0\,1^n\,0$ $(n = 0, 1, \ldots)$ is given as an input. Then, T_{parity} halts in the accepting state q_{acc} if and only if n is even, and all the read symbols are complemented. The computing process for the input string 0110 is as follows: $q_0 0110 \vdash 1 q_1 110 \vdash 10 q_2 10 \vdash 100 q_1 0 \vdash 10 q_{\text{acc}} 01$. $\square$

It is known that any (irreversible) TM can be simulated by an RTM without generating garbage information [1]. Hence, RTMs are computationally universal. In the following, we consider only 2-symbol RTMs with a rightward-infinite tape, since any RTM can be converted into such an RTM (see, e.g., [8]).

5.2 Functional Modules Composed of RLEM 4-31 for RTMs

We compose two kinds of functional modules out of RLEM 4-31. They are a memory cell and a q_i-module (or state module) for 2-symbol RTMs [13].

A *memory cell* simulates one tape square and the movements of the head. It is formulated as a 4-state RSM, since it keeps a tape symbol 0 or 1, and the information whether the head is on this tape square or not (its precise formulation as a 4-state RSM is found in [13]). It has ten input symbols listed in Table 1, and ten output symbols corresponding to the input symbols (for example, for the input symbol W0, there is an output symbol W0′).

A memory cell is composed of nine RLEM 4-31's as shown in Fig. 20**(a)**. Its top RLEM keeps a tape symbol. Namely, if the tape symbol is 0 (1, respectively), then the top RLEM is in the state 0 (1). The position of the head is kept by the remaining eight RLEMs. If all the eight RLEMs are in the state 1, then the head is on this square. If they are in the state 0, the head is not on this square. Figure 20**(a)** shows that the tape symbol is 1 and the head is not on this square. The eight RLEMs also process the instruction/response signals in Table 1 properly.

Table 1. Ten kinds of input symbols of the memory cell [13]

Symbol	Instruction/Response	Meaning
W0	Write 0	Instruction of writing the tape symbol 0 at the head position. By this instruction, read operation is also performed
W1	Write 1	Instruction of writing the tape symbol 1 at the head position. By this instruction, read operation is also performed
R0	Read 0	Response signal telling the read symbol at the head is 0
R1	Read 1	Response signal telling the read symbol at the head is 1
SL	Shift-left	Instruction of shift-left operation
SLI	Shift-left immediate	Instruction of placing the head on this cell by shifting left
SLc	Shift-left completed	Response (completion) signal of shift-left operation
SR	Shift-right	Instruction of shift-right operation
SRI	Shift-right immediate	Instruction of placing the head on this cell by shifting right
SRc	Shift-right completed	Response (completion) signal of shift-right operation

Connecting an infinite number of memory cells in a row, we obtain a tape unit. The read, write and head-shift operations are executed by giving an instruction signal to an input port of the leftmost memory cell. Its response signal comes out from an output port of the leftmost one. Note that a read operation is performed by the Write 0 (W0) instruction. Thus it is a destructive read. The details of how these operations are performed in this circuit is found in [13].

A *q_i-module* corresponds to the state q_i of an RTM. It has five RLEM 4-31's as shown in Fig. 20**(b)**, which is for a shift-right state. Since a module for a shift-left state is similar, we explain only the case of shift-right below. It is further decomposed into a *pre-q_i-submodule* (left half), and a *post-q_i-submodule* (right-half). Note that only the post-q_i-submodule is used for the initial state, while only the pre-q_i-submodule is used for a halting state.

The pre-q_i-submodule is activated by giving a signal from the port $0Rq_i$ or $1Rq_i$ in the figure. Then, a write operation and a shift-right operation that should be done before going to q_i are performed. Namely, it first sends an instruction W0 or W1 to write a tape symbol 0 or 1. Here we assume the previous tape symbol under scan is 0. Then it receives a response signal R0 (i.e., the read symbol is 0). Next, it sends a shift-right instruction SR, and receives a completion signal SRc. After that, it goes to the state q_i, and activates the post-q_i-submodule. The red particle in the figure shows that it has become the state q_i.

In the post-q_i-submodule, a read operation is performed, which should be executed just after becoming the state q_i. By giving an instruction W0, a destructive read operation of a tape symbol is executed. Thus it receives a response signal R0 or R1, and finally a signal goes out from the port q_i0 or q_i1.

Let Q be the set of states of an RTM. For each $q_i \in Q$ prepare a q_i-module and place them in a row. If there is a quintuple $[q_i, s, t, d, q_j]$, then connect the port q_is of the post-q_i-submodule to the port tdq_j of the pre-q_j-submodule. By this, the operation of the quintuple $[q_i, s, t, d, q_j]$ is correctly simulated. Note that such a connection is done in a one-to-one manner (i.e., connection lines neither branch nor merge) since the TM is deterministic and reversible. In this way, a circuit module for the finite state control of the RTM is composed.

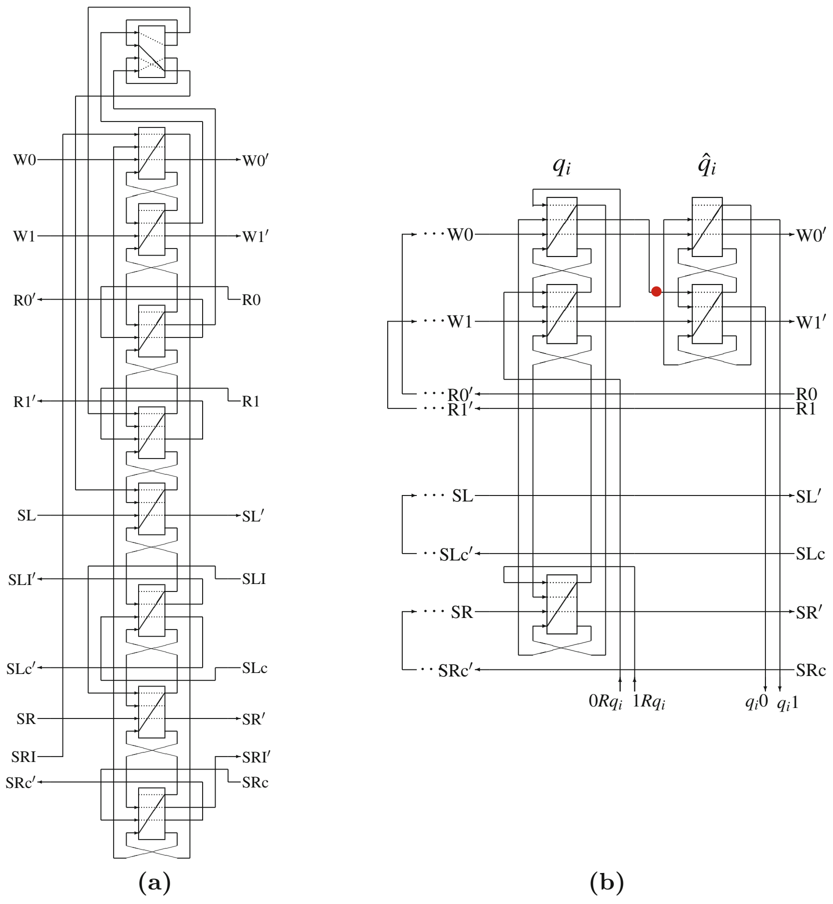

Fig. 20. **(a)** A memory cell, and **(b)** a q_i-module composed of RLEM 4-31 for 2-symbol RTMs [13]. The q_i-module consists of a *pre-q_i-submodule* (left half), and a *post-q_i-submodule* (right-half)

5.3 Constructing RTMs in ETPCA 0347

Assembling memory cells and q_i-modules by the method described in Sect. 5.2, we can compose any 2-symbol RTM. Figure 21 shows the whole circuit that simulates T_{parity} given in Example 1. If a signal is given to the "Begin" port, it starts to compute. Its answer will be obtained at "Accept" or "Reject" port.

Putting copies of the pattern of RLEM 4-31 given in Fig. 18 at the positions corresponding to the RLEMs in Fig. 21, and connecting them appropriately, we have a complete configuration of ETPCA 0347 that simulates T_{parity} in Example 1. Figure 22 shows the configuration simulated on Golly. Giving a glider to "Begin" port, its computation starts. Whole computing processes of RTMs embedded in ETPCA 0347 can be seen on Golly using its emulator [7].

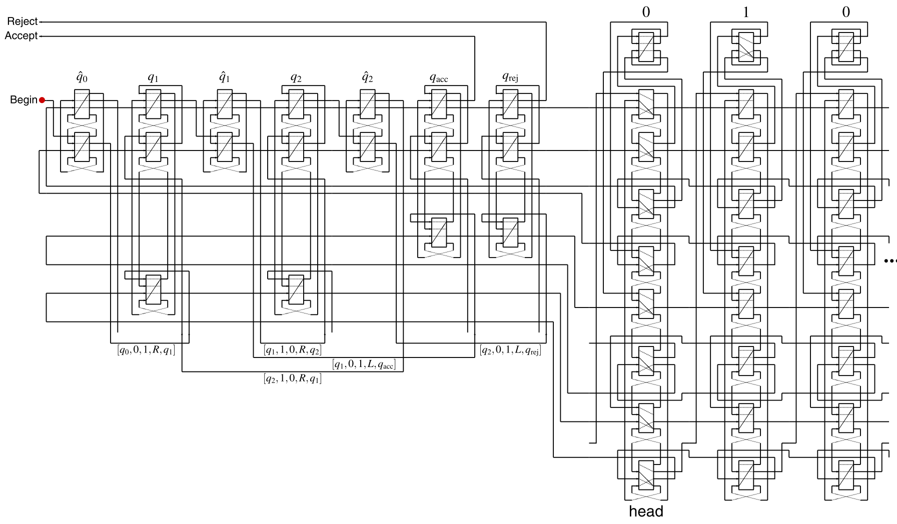

Fig. 21. A circuit composed of RLEM 4-31 that simulates the RTM T_{parity} [13]

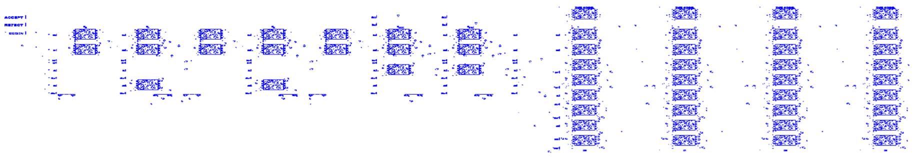

Fig. 22. A configuration of ETPCA 0347 in Golly that simulates the circuit of Fig. 21

6 Concluding Remarks

We studied the problem of how we can find a method of constructing reversible computers from a simple reversible microscopic law. We put several conceptual levels properly on the construction pathway as in Fig. 1. By this, the problem was decomposed into several subproblems, and it became feasible to solve it. Here we saw that it is possible to compose RTMs even from the extremely simple local function of reversible ETPCA 0347 using RLEM 4-31 in the intermediate step of the pathway.

Only the specific ETPCA 0347 is considered here, but there are many other ETPCAs. In [10], this problem is studied using reversible ETPCA 0137. There, observed phenomena in the level 2 and composing technics of RLEM 4-31 in the level 3 are very different. However, also in this case, the systematic and hierarchical method shown in Fig. 1 can be applied, and works effectively.

References

1. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**, 525–532 (1973). https://doi.org/10.1147/rd.176.0525
2. Fredkin, E., Toffoli, T.: Conservative logic. Int. J. Theoret. Phys. **21**, 219–253 (1982). https://doi.org/10.1007/BF01857727
3. Imai, K., Morita, K.: A computation-universal two-dimensional 8-state triangular reversible cellular automaton. Theoret. Comput. Sci. **231**, 181–191 (2000). https://doi.org/10.1016/S0304-3975(99)00099--7
4. Kari, J.: Reversibility and surjectivity problems of cellular automata. J. Comput. Syst. Sci. **48**, 149–182 (1994). https://doi.org/10.1016/S0022-0000(05)80025-X
5. Lee, J., Peper, F., Adachi, S., Morita, K.: An asynchronous cellular automaton implementing 2-state 2-input 2-output reversed-twin reversible elements. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 67–76. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79992-4_9
6. Morita, K.: Finding a pathway from reversible microscopic laws to reversible computers. Int. J. Unconventional Comput. **13**, 203–213 (2017)
7. Morita, K.: Reversible world : Data set for simulating a reversible elementary triangular partitioned cellular automaton on Golly. Hiroshima University Institutional Repository. http://ir.lib.hiroshima-u.ac.jp/00042655 (2017)
8. Morita, K.: Theory of Reversible Computing. Springer, Tokyo (2017). https://doi.org/10.1007/978-4-431-56606-9
9. Morita, K.: A universal non-conservative reversible elementary triangular partitioned cellular automaton that shows complex behavior. Nat. Comput. **18**(3), 413–428 (2017). https://doi.org/10.1007/s11047-017-9655-9
10. Morita, K.: Constructing reversible Turing machines in a reversible and conservative elementary triangular cellular automaton. J. Automata, Lang. Combinatorics (to appear)
11. Morita, K., Harao, M.: Computation universality of one-dimensional reversible (injective) cellular automata. Trans. IEICE **E72**, 758–762 (1989). http://ir.lib.hiroshima-u.ac.jp/00048449
12. Morita, K., Ogiro, T., Alhazov, A., Tanizawa, T.: Non-degenerate 2-state reversible logic elements with three or more symbols are all universal. J. Multiple-Valued Logic Soft Comput. **18**, 37–54 (2012)
13. Morita, Kenichi, Suyama, Rei: Compact realization of reversible turing machines by 2-state reversible logic elements. In: Ibarra, Oscar H.., Kari, Lila, Kopecki, Steffen (eds.) UCNC 2014. LNCS, vol. 8553, pp. 280–292. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08123-6_23
14. Mukai, Y., Ogiro, T., Morita, K.: Universality problems on reversible logic elements with 1-bit memory. Int. J. Unconventional Comput. **10**, 353–373 (2014)
15. Trevorrow, A., Rokicki, T., Hutton, T., et al.: Golly: an open source, cross-platform application for exploring Conway's Game of Life and other cellular automata (2005). http://golly.sourceforge.net/