The Logic in Computer Science Column

BY

Yuri Gurevich

Computer Science & Engineering
University of Michigan, Ann Arbor, Michigan, USA
gurevich@umich.edu

Making Reversible Computing Machines in a Reversible Cellular Space

Kenichi Morita
Hiroshima University, Higashi-Hiroshima 739-8527, Japan
Currently Professor Emeritus of Hiroshima University
km@hiroshima-u.ac.jp

Abstract

Reversible computing is a study that investigates the problem of how computing is effectively performed in a reversible world. Since physical reversibility is one of the fundamental microscopic laws of nature, it is important to clarify how computing machines are realized utilizing a reversible law directly. In this survey/tutorial paper, we investigate this problem using a reversible cellular automaton as a reversible environment, and search for a new way of constructing reversible Turing machines (RTMs), a model of a reversible computer, in it. That is to find a good pathway from a reversible microscopic law to reversible computers. When doing so, it is convenient to assume several conceptual levels on the pathway, by which the problem is decomposed into subproblems. In the middle level on the pathway we use a reversible logic element with 1-bit memory (RLEM), rather than a reversible logic gate, as a logical primitive. By these methods, we see that RTMs can be implemented systematically even in a space that obeys a very simple reversible microscopic law.

1 Introduction

A reversible computing machine is a system having a "backward deterministic" property. That is to say, every computational state of the machine has at most one predecessor state. Though its definition is thus simple, it has a close relation to the physical reversibility, one of the fundamental microscopic laws in physics [1, 8]. Therefore, it is important to know how reversible machines are realized by utilizing reversible microscopic phenomena.

So far many kinds of reversible computing models have been proposed and studied. One method of showing that a reversible computing model has a sufficient computing power is to simulate an irreversible version of the model by a

reversible one, i.e., "reversifying" [4] the irreversible system. By this method, computational/logical universality of various reversible models have been shown. Reversifying techniques have been applied, for example, to Turing machines [1], logic elements and circuits [3, 26], two-counter machines [12], two-way finite automata [7], two-way multihead finite automata [14], cellular automata [25], and so on. Once universality of a reversible computing model is established by this method, universality of another reversible model can be shown by simulating the former by the latter. For example, it has been shown that a universal reversible logic gate and its circuits are simulated by a simple reversible 2D cellular automaton [11], and that a reversible Turing machine can be simulated by a reversible 1D cellular automaton [19]. In this way, it turned out that, in many computing models, computing powers do not decrease even if the reversibility constraint is added.

Besides the study of individual reversible computing machines, it is also important to investigate how these machines can be efficiently realized in a space that obeys a simple reversible law. In other words, it is to investigate the problems of how simple reversible primitive operations can be that support universal computation, and how reversible macroscopic systems can be realized from a reversible microscopic law. When we try to implement reversible machines in such a simple environment, it is convenient to consider several implementation levels ranging from a microscopic level to a macroscopic one as shown in Fig. 1. By this, the problem is decomposed into several simpler subproblems. In the bottom level, i.e., Level 1, there is a simple microscopic reversible law of evolution, which corresponds to a microscopic physical law. In Level 2, various phenomena that emerge from the reversible microscopic law can be observed. In Level 3, we implement suitable reversible logic elements using the observed phenomena. In Level 4, combining the reversible logic elements, functional modules for reversible computers are composed. In the top level, i.e., Level 5, a reversible computing machine is systematically constructed by assembling the reversible functional modules.

Whether we can successfully find a pathway from a reversible microscopic law to reversible computers firstly depends on the choice of the reversible microscopic law in Level 1. In this paper, we use a reversible cellular automaton for it as a thought experiment. It is a reversible elementary square partitioned cellular automaton (ESPCA) with a hexadecimal identification number "01caef"' [17], which is denoted by P_0 for short in this paper. It is described by only six local transition rules, and thus very simple. Though the reversible cellular automaton used here is an artificial model, and its physical realizability in the nano-scale level is not known at present, they will give new vistas in reversible computing. In particular, we shall see that even from very simple local rules, useful phenomena that can be used for composing reversible machines are found in Level 2.

It also depends on the choice of reversible logic elements in Level 3. Here, we

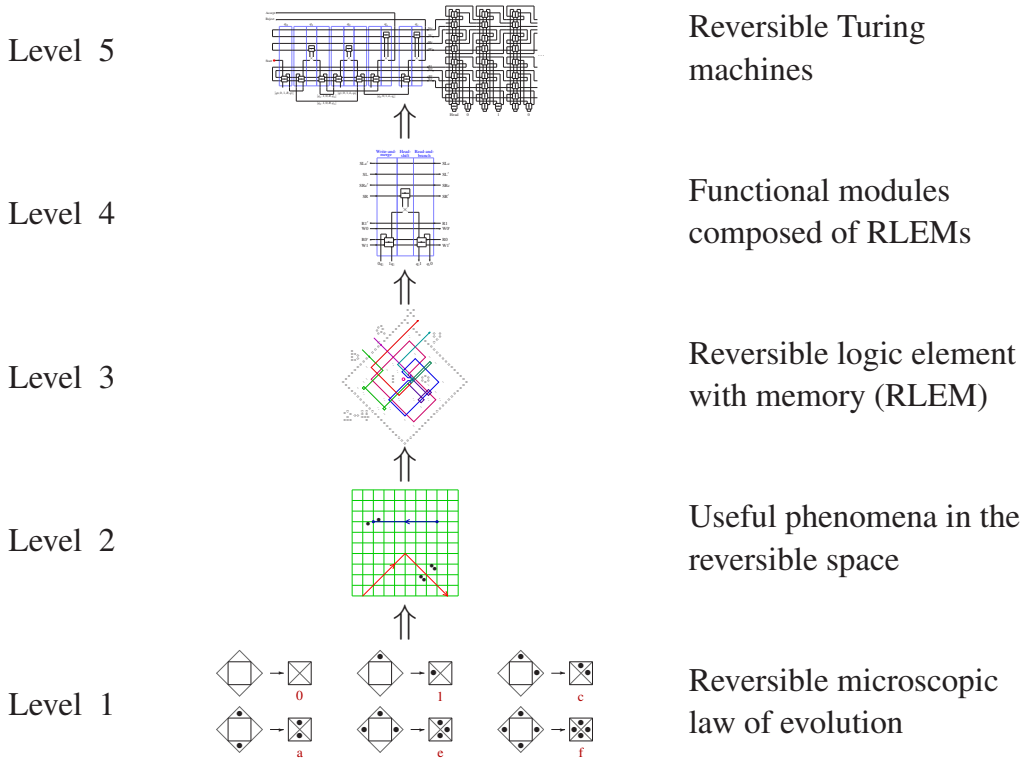


Figure 1: A pathway from a reversible microscopic law to reversible computers

use a reversible logic element with 1-bit memory (RLEM) rather than a reversible logic gate. We shall see that RLEMs can be implemented using a small number of phenomena found in Level 2. In addition, construction of reversible computing machines in the upper levels is greatly simplified.

In Levels 4 and 5, a reversible Turing machine (RTM), an abstract model of a reversible computer, is constructed. To do so, in Level 4, functional modules are composed out of RLEMs. Then, in Level 5, RTMs are systematically constructed by assembling the modules.

The contents of the following sections are as follows. In Sect. 2 reversible Turing machines are defined, and computational universality results of their restricted subclasses are surveyed. In Sect. 3 reversible logic elements with memory (RLEM) are given, and a construction method of RTMs using a particular RLEM called a rotary element (RE) is explained. In Sect. 4 a very simple 2D cellular automaton called an elementary square partitioned cellular automaton (ESPCA) is introduced, and useful phenomena in the particular reversible ESPCA P_0 are explored. Using them, an RE is implemented in P_0, and then RTMs are realized as configurations of P_0. Sect. 5 gives concluding remarks.

2 Reversible Turing Machine (RTM)

A reversible Turing machine (RTM) is a standard model in the theory of reversible computing. Lecerf [9] first investigated RTMs, and showed unsolvability of the halting problems and some related problems. Bennett [1] studied them from the viewpoint of thermodynamics of computing, and showed that any irreversible TM can be converted into an equivalent RTM.

2.1 Definitions and examples

A one-tape Turing machine (TM) consists of a finite control, a read-write head, and a two-way infinite tape divided into squares in which symbols are written. Formal definition of a TM is as follows.

Definition 2.1. A *one-tape Turing machine (TM)* is defined by

$$T = (Q, S, q_0, F, s_0, \delta),$$

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 special *blank symbol* ($s_0 \in S$). Here, δ is a move relation, which is a subset of $(Q \times S \times S \times \{L, N, R\} \times Q)$. The symbols "$L$", "$N$", and "$R$" are *shift directions* of the head, which stand for "left-shift", "no-shift", and "right-shift", respectively. Each element of δ is a *quintuple* of the form $[p, s, s', d, q]$, which is called a *rule* of T. It means if T reads the symbol s in the state p, then write s', shift the head to the direction d, and go to the state q. We assume each state $q_f \in F$ is a *halting state*, i.e., there is no quintuple of the form $[q_f, s, s', d, q]$ in δ. In this paper, we assume T is deterministic. Hence, 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, the relation $(p_1 = p_2) \Rightarrow (s_1 \neq s_2)$ holds.

Reversibility of a TM is defined as below.

Definition 2.2. Let $T = (Q, S, q_0, F, s_0, \delta)$ be a TM. We call T a *reversible TM* (RTM), 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.

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

It means that for any pair of distinct rules, if the next states are the same, then the shift directions are the same, and the written symbols are different. The above is called the *reversibility condition* for TMs.

Note that, in [1], RTMs are defined in a quadruple form, where read-write rules and head-shift rules are separated. This formulation is useful when composing an "inverse" RTM that undoes the computation performed by a given RTM. However, here, we employ the quintuple formulation, since the number of rules for defining an RTM in the quintuple form is about a half of that for defining an RTM in the quadruple form. See Sect. 5.1.3 of [15] for a conversion method between these two forms.

An instantaneous description (ID) of a TM is an expression to describe its computational configuration.

Definition 2.3. Let $T = (Q, S, q_0, F, s_0, \delta)$ be a one-tape TM. We assume $Q \cap S = \emptyset$. An *instantaneous description* (ID) of T is a string of the form $\alpha q \beta$ where $q \in Q$ and $\alpha, \beta \in S^*$. Let λ denote the empty string. The ID $\alpha q \beta$ describes the *computational configuration* of T such that the content of the tape is $\alpha\beta$ (the remaining part of the tape contains only blank symbols), and T is reading the leftmost symbol of β (if $\beta \neq \lambda$) or s_0 (if $\beta = \lambda$) in the state q. An ID $\alpha q \beta$ is called a *standard form ID* if $\alpha \in (S - \{s_0\})S^* \cup \{\lambda\}$, and $\beta \in S^*(S - \{s_0\}) \cup \{\lambda\}$. Namely, a standard form ID is obtained from a general ID by removing superfluous blank symbols from the left and the right ends. An ID $\alpha q_0 \beta$ is called an *initial ID*. An ID $\alpha q \beta$ is called a *final ID* if $q \in F$.

The transition relation among standard form IDs of T is denoted by $\vdash_T$. Let $\alpha q \beta$ and $\alpha' q' \beta'$ be two standard form IDs. If $\alpha' q' \beta'$ is obtained from $\alpha q \beta$ by applying a rule in δ of T, then we write $\alpha q \beta \vdash_T \alpha' q' \beta'$, and say that T goes to the computational configuration $\alpha' q' \beta'$ from $\alpha q \beta$ in one step. For example, if $[q, s, s', R, q'] \in \delta$, $\alpha \in (S - \{s_0\})S^*$, and $\beta \in S^*(S - \{s_0\})$, then $\alpha q s \beta \vdash_T \alpha s' q' \beta$. Though the relation $\vdash_T$ is conceptually straightforward one, its formal definition is slightly complex, since only standard form IDs are considered, and thus we have to deal with many cases. Hence, its definition is omitted here (see Sect. 5.1.1.3 of [15] for its precise definition).

The reflexive and transitive closure of $\vdash_T$ is denoted by $\vdash_T^*$. The transitive closure is denoted by $\vdash_T^+$. The relation of n-step transition is denoted by $\vdash_T^n$. Let γ be a standard form ID of T. We say γ is a *halting ID*, if there is no ID γ' such that $\gamma \vdash_T \gamma'$. Let $\alpha_i, \beta_i \in S^*$, and $p_i \in Q$ $(n \in \mathbb{N}, i = 0, 1, \ldots, n)$. We say $\alpha_0 p_0 \beta_0 \vdash_T \alpha_1 p_1 \beta_1 \vdash_T \cdots \vdash_T \alpha_n p_n \beta_n$ (or $\alpha_0 p_0 \beta_0 \vdash_T^* \alpha_n p_n \beta_n$) is a *complete computing process* of T starting from $\alpha_0 p_0 \beta_0$, if $\alpha_0 p_0 \beta_0$ is an initial ID (i.e., $p_0 = q_0$), and $\alpha_n p_n \beta_n$ is a halting ID.

We give two examples of RTMs. In the following sections, they are constructed using reversible logic element with memory (RLEM), and then implemented in a simple reversible cellular automaton.

Example 2.1. An RTM T_{parity} defined below is a very simple example.

$$T_{\text{parity}} = (Q_{\text{parity}}, \{0, 1\}, q_0, \{q_{\text{a}}\}, 0, \delta_{\text{parity}})$$

Here, $Q_{\text{parity}} = \{q_0, q_1, q_2, q_{\text{a}}, q_{\text{r}}\}$, and δ_{parity} are given below.

$$\begin{aligned}\delta_{\text{parity}} = \{\ & [q_0, 0, 1, R, q_1],\ [q_1, 0, 1, L, q_{\text{a}}],\ [q_1, 1, 0, R, q_2], \\ & [q_2, 0, 1, L, q_{\text{r}}],\ [q_2, 1, 0, R, q_1]\ \}\end{aligned}$$

It is easy to see that T_{parity} is reversible. Consider the pair of rules $[q_0, 0, 1, R, q_1]$ and $[q_2, 1, 0, R, q_1]$. The next states in these rules are the same (i.e., q_1). We can see the shift directions in them are the same (i.e., R), and the written symbols are different (i.e., 1 and 0). Thus the pair satisfies the reversibility condition in Definition 2.2. No other pair of distinct rules have the same next state. Therefore T_{parity} is reversible. Complete computing processes starting from the IDs q_0011 and q_00111 are as follows.

$$\begin{aligned}&q_0011 \vdash_{T_{\text{parity}}} 1q_111 \vdash_{T_{\text{parity}}} 10q_21 \vdash_{T_{\text{parity}}} 100q_1 \vdash_{T_{\text{parity}}} 10q_{\text{a}}01 \\ &q_00111 \vdash_{T_{\text{parity}}} 1q_1111 \vdash_{T_{\text{parity}}} 10q_211 \vdash_{T_{\text{parity}}} 100q_11 \vdash_{T_{\text{parity}}} 1000q_2 \vdash_{T_{\text{parity}}} 100q_{\text{r}}01\end{aligned}$$

For a given string 01^n, the RTM T_{parity} tests whether n is even or not. If it is even, T_{parity} halts in the final (accepting) state q_{a}. Otherwise it halts in q_{r}. All the read symbols are complemented.

Example 2.2. An RTM T_{power} is defined by

$$T_{\text{power}} = (Q_{\text{power}}, \{0, 1\}, q_0, \{q_{\text{a}}\}, 0, \delta_{\text{power}}).$$

Here, $Q_{\text{power}} = \{q_0, q_1, \ldots, q_7, q_{\text{a}}, q_{\text{r}}\}$, and δ_{power} are given below.

$$\begin{aligned}\delta_{\text{power}} = \{\ & [q_0, 0, 0, R, q_1],\ [q_1, 0, 0, R, q_2],\ [q_2, 0, 0, L, q_6],\ [q_2, 1, 0, R, q_3], \\ & [q_3, 0, 1, L, q_4],\ [q_3, 1, 1, R, q_3],\ [q_4, 0, 0, L, q_7],\ [q_4, 1, 0, L, q_5], \\ & [q_5, 0, 1, R, q_2],\ [q_5, 1, 1, L, q_5],\ [q_6, 0, 0, L, q_{\text{r}}],\ [q_6, 1, 1, R, q_1], \\ & [q_7, 0, 0, L, q_{\text{a}}],\ [q_7, 1, 1, L, q_{\text{r}}]\ \}\end{aligned}$$

It is again easy to see that T_{power} satisfies the reversibility condition. Complete computing processes starting from $q_0001111$ and $q_000111111$ are as follows.

$$\begin{aligned}&q_0001111 \vdash^{31}_{T_{\text{power}}} 110\, q_{\text{a}}1001 \\ &q_000111111 \vdash^{43}_{T_{\text{power}}} 111\, q_{\text{r}}\, 01011\end{aligned}$$

For a given string 001^n, the RTM T_{power} tests whether n is a power of 2. If it is so, T_{power} halts in the final state q_{a}. Otherwise it halts in q_{r}. It uses a straightforward algorithm that repeatedly divides the unary number n by 2, and checks the remainder at each division. But, note that, T_{power} is carefully designed so that it satisfies the reversibility condition.

2.2 Computational universality of RTMs

Bennett [1] showed that any one-tape irreversible TM can be converted into an equivalent three-tape RTM. Hence, the class of three-tape RTMs is computationally universal.

Theorem 2.1. *For any (irreversible) one-tape TM, we can construct a reversible three-tape RTM that simulates the former and leaves no garbage information on its tape.*

Assume some class of RTMs is known to be computationally universal. If any RTM in this class is simulated by an RTM in another class of RTMs, then the latter class of RTMs is also computationally universal. In this way, computational universality of various subclasses of RTMs can be shown. In particular, it is possible to show the following.

(1) For any RTM with k two-way infinite tapes, we can construct an RTM with k one-way infinite (i.e., rightward infinite) tapes that simulates the former ($k = 1, 2, \ldots$).

(2) For any RTM with k rightward infinite tapes, we can construct an RTM with only one rightward infinite tape that simulates the former ($k = 2, 3, \ldots$).

(3) For any k-symbol RTM with one rightward infinite tape, we can construct a two-symbol RTM with one rightward infinite tape that simulates the former ($k = 3, 4, \ldots$).

In the case of irreversible TMs, it is relatively easy to show the results corresponding to the above (see, e.g., [6] for (1), [5] for (2), and [24] for (3)). However, in the case of RTMs, the simulating TMs should be carefully constructed so that they satisfy the reversibility condition. In addition, the notion of simulation should also be defined properly. These details are found in Sect. 5.3 of [15].

By above, we obtain the following.

Theorem 2.2. *The class of two-symbol RTMs with a rightward infinite tape is computationally universal.*

In the following sections, only two-symbol RTMs with a rightward infinite tape are constructed by reversible logic elements, and then implemented in a simple reversible cellular automaton.

It has been shown that a one-tape many-state RTM can be simulated by a one-tape three-state RTM having many symbols (see Sect. 5.3.5 of [15]). Therefore we have the following.

Theorem 2.3. *The class of three-state RTMs with a rightward infinite tape is computationally universal.*

In the case of irreversible TMs, it is known that a many-state TM can be simulated by a two-state TM [24]. Hence the class of two-state TMs is universal. However, it is unknown whether the class of two-state RTMs is universal.

A *universal Turing machine* (UTM) is one that can simulate any TM. Let UTM(m,n) denote an m-state n-symbol UTM. It is known that various kinds of UTMs with very small m and n exist. For example, Rogozhin [23] gave UTM(4,6), and Neary and Woods [22] gave UTM(6,4), which simulate 2-tag systems and bi-tag systems, respectively. These UTMs have the smallest value of $m \times n$ among the ones so far found.

It is, of course, possible to have a *universal reversible Turing machine* (URTM) by reversifying a UTM using the method of Bennett (Theorem 2.1) and then converting it into a one-tape RTM. However, if we do so, m and n become very large. In the case of URTM(m,n) with m states and n symbols, a method of simulating cyclic tag systems [2] was used to have ones with small m and n (Sect. 7.3 of [15]). Among them, URTM(10,8) has the smallest value of $m \times n$.

3 Reversible Logic Element with Memory (RLEM)

A *reversible logic element with memory* (RLEM) [13] is a kind of a reversible finite automaton having output symbols as well as input symbols, which is also called a reversible sequential machine of Mealy type. In the following, we use RLEMs rather than reversible logic gates for composing RTMs.

Definition 3.1. A *sequential machine* (SM) 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 $\delta : Q \times \Sigma \to Q \times \Gamma$ is a move function (see Fig. 2 **(a)**). If δ is injective, it is called a *reversible sequential machine* (RSM).

To use an SM as a logic element, we interpret it as the one with decoded input/output ports (Fig. 2 **(b)**), i.e., for each input symbol, there is a unique input port to which a signal (or a particle) is given. It is also the case for the output symbols. Therefore, signals should not be given to two or more input ports at the same time.

An RLEM is an RSM that satisfies $|\Sigma| = |\Gamma|$. When connecting many RLEMs to form an RLEM-circuit, each output port of an RLEM can be connected to at most one input port of another (or may be the same) RLEM. Furthermore, two or more output ports should not be connected to one input port. Therefore, neither branching (i.e., fan-out of an output) nor merging of signal lines is permitted. See Sect. 3.5.1 of [15] for the precise definition of an RLEM-circuit.

Among RLEMs, two-state RLEMs are particularly important, since they are simple yet powerful (see Sect. 3.4). In the following, we use a specific RLEM, a rotary element (RE), to compose RTMs. This is because the operation of RE is intuitively easy to understand, and RTMs can be constructed by it very simply.

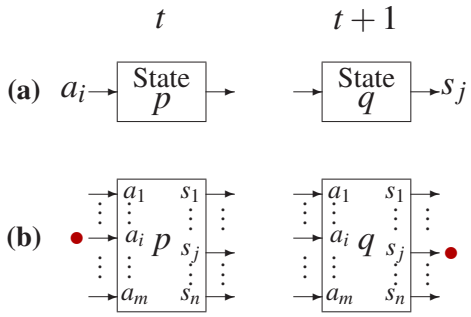


Figure 2: **(a)** A sequential machine with $\delta(p, a_i) = (q, s_j)$, and **(b)** an interpretation of it as a module having decoded input ports and output ports

3.1 Rotary element (RE), a typical RLEM

A *rotary element* (RE) [13] is a two-state RLEM that has four input ports and four output ports, and is depicted as in Fig. 3. Intuitively, an RE has a rotatable bar inside, and an incoming signal is controlled by the bar. It takes either of the two states, state V or state H, depending on the direction of the bar. If the direction of a coming signal is parallel to the bar, the signal goes straight ahead, and the state does not change (Fig. 4 **(a)**). If the direction of a coming signal is orthogonal to the bar, the signal turns right, and the state changes (Fig. 4 **(b)**).

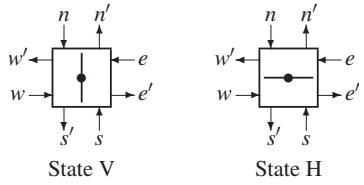


Figure 3: Two states of a rotary element (RE)

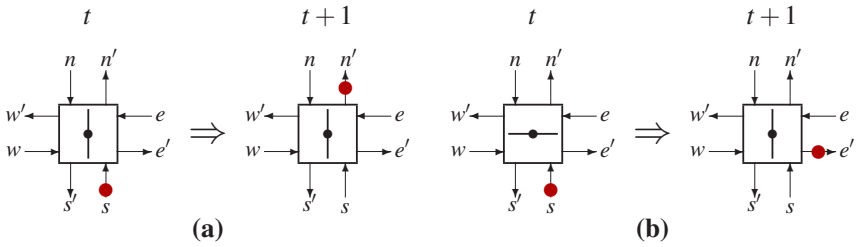


Figure 4: Operations of an RE. **(a)** Parallel case, and **(b)** orthogonal case

3.2 Constructing reversible sequential machines using REs

We can construct any RSM using only REs. To do so, we introduce a circuit module called an RE-column. RSMs are composed of it systematically.

3.2.1 RE-column, a module for building RSMs

An *RE-column* of degree n is shown in Fig. 5, which has $n+1$ REs. We assume, in a resting state, it is in the state **(a)** or **(b)** of Fig. 5, where all the REs except the bottom are in the state V. It has $2n$ input ports $a_1, \ldots, a_n, b_1, \ldots, b_n$, and $2n$ output ports $s_1, \ldots, s_n, t_1, \ldots, t_n$. If a signal is given to one of the input ports, the module will take a state other than those of **(a)** and **(b)**. However, as we shall see, the module will become again the state **(a)** or **(b)** when the signal goes out from it. Therefore, an RE-column behaves as if it is a two-state RSM. That is to say, the states **(a)** and **(b)** are macroscopic states 0 and 1 of the RE-column.

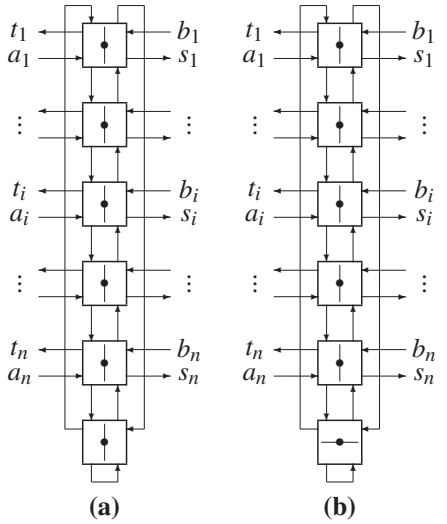


Figure 5: RE-column of degree n. **(a)** State 0, and **(b)** state 1

Table 1: The move function of an RE-column of degree n. Here, $i \in \{1, \ldots, n\}$

	Input	
Present state	a_i	b_i
0	$0\, s_i$	$1\, s_i$
1	$0\, t_i$	$1\, t_i$

The move function of the RE-column as a two-state RSM is shown in Table 1. In the following, we examine how the circuit works for the four cases of state-input pairs: $(0, a_i)$, $(1, a_i)$, $(0, b_i)$, and $(1, b_i)$, where $i \in \{1, \ldots, n\}$.

First, consider the case where the state is 0 (Fig. 5 **(a)**) and a signal is given to a_i. By the signal from a_i, the i-th RE changes its state from V to H. Then the signal moves downward through the $(n - i + 1)$ REs. At the bottom of the column the signal makes a U-turn, and goes upward through the $(n - i + 1)$ REs. At the i-th RE, the signal turns right and changes the RE's state from H to V. Finally the signal goes out from the port s_i. In this case the RE-column keeps the state 0.

Second, consider the case where the state is 1 (Fig. 5 **(b)**) and a signal is given to a_i. As in the first case, the signal sets the i-th RE to the state H, and then moves downward. At the bottom RE, the signal makes a right-turn, and changes the state of the RE to V. The signal goes upward along the left vertical line, and reaches the north input of the top RE. It moves downward through the $(i - 1)$ REs, and makes a right-turn at the i-th RE, restoring the RE's state to V. Finally the signal goes out from the port t_i. In this case the RE-column changes the state from 1 to 0.

Third, consider the case where the state is 0 and a signal is given to b_i. The signal sets the i-th RE to the state H, and moves upward through the $(i - 1)$ REs. Then the signal goes downward along the right vertical line, and reaches the east input of the bottom RE. It changes the state of the bottom RE to H, and moves upward through the $(n - i)$ REs. The signal makes a right-turn at the i-th RE, and restores its state to V. Finally the signal goes out from the port s_i. In this case the RE-column changes the state from 0 to 1.

Fourth, consider the case where the state is 1 and a signal is given to b_i. As in the third case, the signal sets the i-th RE to the state H, and reaches the east input of the bottom RE. The signal goes out from the west output of the bottom RE without changing its state. It moves upward along the left vertical line, and reaches the north input of the top RE. It makes a right-turn at the i-th RE, restoring the RE's state to V. Finally the signal goes out from the port t_i. In this case the RE-column keeps the state 1.

3.2.2 Composing RSMs using RE-columns

We can systematically compose any RSM out of RE-columns. The composing method is explained by the following example of an RSM M_0.

$$M_0 = (\{q_1, q_2, q_3\}, \{c_1, c_2, c_3\}, \{d_1, d_2, d_3\}, \delta_0)$$

The move function δ_0 is given in Table 2.

Fig. 6 shows the circuit that simulates M_0. It consists of three RE-columns of degree 3. The j-th RE-column corresponds to the j-th state q_j of M_0. The i-th row except the bottom row corresponds to the input symbol c_i and the output symbol d_i. If the state of M_0 is q_j, then the state of the j-th RE-column is set to 1, while the other RE-columns are set to 0. Fig. 6 shows that M_0 is in the state q_3.

For example, assume an input signal is given to the port c_2. Since the first two RE-columns are in the state 0, the signal goes rightward through these RE-columns without changing their states. At the third RE-column, the signal changes the RE-column's state from 1 to 0, and then comes out from the west output port of the second RE, which is labeled by q_3c_2. This port is connected to the east input port of the third RE of the second RE-column labeled by q_2d_3. By this, the state of the second RE-column changes from 0 to 1. The signal appears from the east output port of the third RE in the second RE-column. Since the third RE-column is now in the state 0, the signal finally goes out from the port d_3. By above, the operation $\delta_0(q_3, c_2) = (q_2, d_3)$ is simulated. Other cases are similar to this case.

Table 2: The move function δ_0 of an RSM M_0

	Input		
Present state	c_1	c_2	c_3
q_1	$q_2\, d_2$	$q_3\, d_1$	$q_1\, d_2$
q_2	$q_3\, d_2$	$q_2\, d_1$	$q_1\, d_3$
q_3	$q_3\, d_3$	$q_2\, d_3$	$q_1\, d_1$

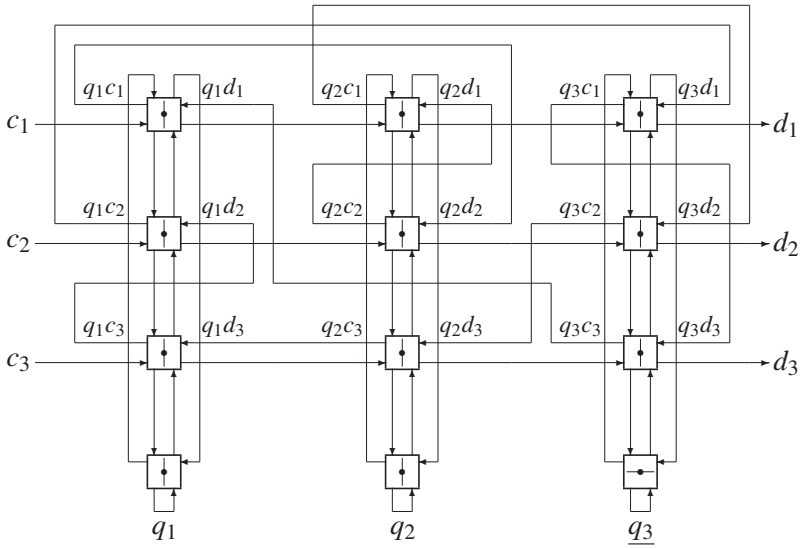


Figure 6: An RSM M_0 composed only of RE [15]

Generally, for any given RSM $M = (\{q_1, \ldots, q_n\}, \{c_1, \ldots, c_l\}, \{d_1, \ldots, d_m\}, \delta)$, we can construct a circuit composed of RE-columns that simulates M in the following way. First prepare n RE-columns of degree $r = \max\{l, m\}$, and connect the s_i output of the j-th RE-column to the a_i input of the $(j+1)$-st RE-column $(i \in \{1, \ldots, r\}, j \in \{1, \ldots, n-1\})$. Also connect c_i to a_i of the first RE-column

($i \in \{1, \ldots, l\}$), and s_i of the n-th RE-column to d_i ($i \in \{1, \ldots, m\}$). For all q_h, q_j, c_i, and d_k, if $\delta(q_h, c_i) = (q_j, d_k)$, then connect the west output of the RE at (i, h) to the east input of the RE at (k, j). By this, M is correctly simulated.

3.3 Constructing reversible Turing machines using REs

Using only REs, we can compose any two-symbol RTM with a rightward infinite tape. A composing method was first given in [13]. Then it was revised in [15], and it is further revised here. An RTM is constructed by assembling two kinds of functional modules. They are a tape cell module and a state module. Note that the tape cell module uses an RE-column as a submodule.

3.3.1 Tape cell module

A *tape cell module* is a circuit shown in Fig. 7. It simulates one tape square of an RTM. Connecting infinite number of copies of it, a tape unit is obtained as shown in the right part of Fig. 12.

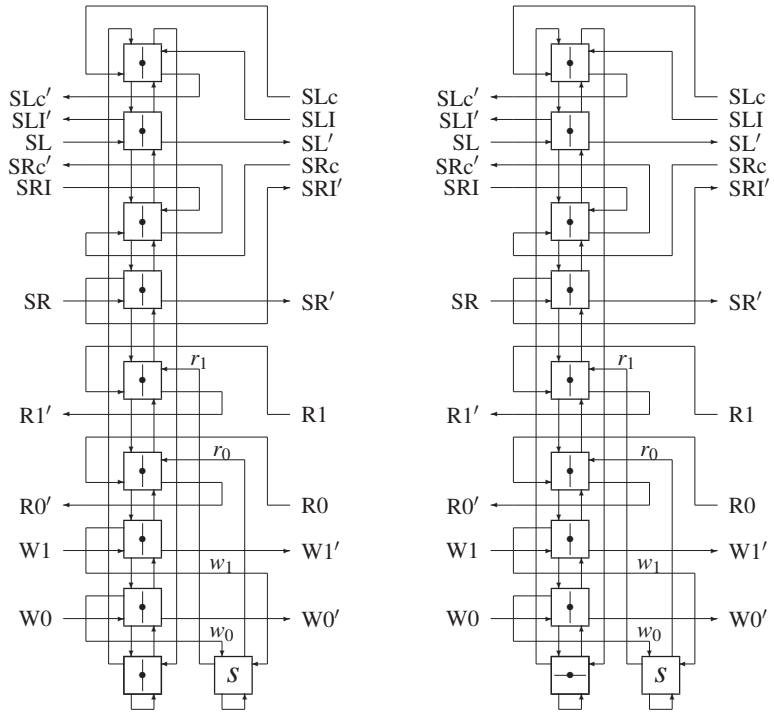


Figure 7: Tape cell module for two-symbol RTMs. The state **(a)** shows that the head is not on this cell, and **(b)** shows that the head is on this cell

The tape cell keeps the information whether the head of the RTM is on this cell or not in its left part, which is an RE-column of degree 8 (Fig. 5). If the RE-column is in the state 0 (i.e., its bottom RE is in the state V), then the head is not here (Fig. 7 **(a)**). If it is in the state 1 (i.e., its bottom RE is in the state H), then the head is here (Fig. 7 **(b)**).

A tape symbol $s \in \{0, 1\}$ is stored in the RE indicated by s in Fig. 7, where 0 and 1 are represented by the states V and H, respectively. The right part of the tape cell is, in fact, a one-bit memory (Fig. 8) having the move function given in Table 3. It has two input ports w_0 and w_1, and two output ports r_0 and r_1. Assume the present state is s ($\in \{0, 1\}$). If a signal is given to w_t ($t \in \{0, 1\}$), then the new symbol t is written in it, and the old symbol s is read-out from the output port r_s. Thus, a write operation always accompanies a read operation.

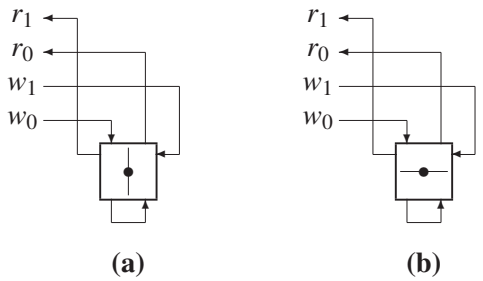


Figure 8: One-bit memory for a tape cell module. **(a)** State 0, and **(b)** state 1

Table 3: The move function of the one-bit memory given in Fig. 8

	Input	
Present state	w_0	w_1
0	$0\,r_0$	$1\,r_0$
1	$0\,r_1$	$1\,r_1$

The tape cell module has ten input ports corresponding to ten kinds of input symbols listed in Table 4. They are interpreted as instructions to the tape unit or response signals to the finite control of an RTM. For each input symbol, there is a corresponding output symbol, which is indicated by the symbol with $'$, and thus the tape cell has ten input ports and ten output ports. Making an infinite number of copies of it, and connecting them to form a rightward infinite array, we can obtain a tape unit for the RTM. To the left of the tape unit a finite control of an RTM will be connected. We assume there is only one tape cell whose RE-column is in the state 1 in the initial setting, and thus there is only one head. Giving a signal to the tape unit, read/write and head-shift operations are performed.

Table 4: Ten kinds of symbols for the tape cell module and their meanings [15]

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

First, consider the case where the head is not on this tape cell (Fig. 7 **(a)**). Since its RE-column is in the state 0, a signal from the input port W0, W1, R0, R1, SL, SR, SLc, or SRc simply goes to the output port W0′, W1′, R0′, R1′, SL′, SR′, SLc′, or SRc′, respectively, without changing its state (see Table 1). It means that these signals skip tape cells having no head. Note that processing of a signal SLI or SRI in this case is discussed later.

Second, consider the case where the head is on this cell (Fig. 7 **(b)**). The first subcase is that an input signal Wt ($t \in \{0, 1\}$) is given, which is for writing the tape symbol t in this tape cell. We assume the one-bit memory in its right part is in the state s. The signal changes the state of the RE-column to 0, and appears on the line w_t in Fig. 7 (see Table 1). Then the state of the one-bit memory changes to t, and the signal appears on the line r_s (see Table 3). This signal restores the RE-column to the state 1, and finally goes out from the port Rs'. Hence, the writing operation also performs a reading operation to keep reversibility of the tape cell. The signal Rs' moves leftward through tape cells having no head, and finally reaches the finite control of the RTM. Note that if an RTM needs to read a tape symbol, it is performed by sending a signal to the input port W0 of the tape unit. By this, the tape unit gives a response signal at the output port Rs', and the tape symbol at the head position is cleared to 0. Thus, this is a destructive readout.

The second subcase is that a signal is given to the input port SL of the tape cell with a tape head, which will shift the tape head to the left. This signal changes the RE-column to the state 0, and goes out from the port SLI′ (Table 1). This signal is sent to the left-neighboring tape cell. If the latter tape cell receives an input signal SLI, then it sets the state of the RE-column to 1, and sends an output signal SLc′ to the left. By such a process, shift-left operation is performed correctly.

The third subcase is that a signal is given to the input port SR of the tape cell

with a tape head, which will shift the tape head to the right. The ports SR, SRI, and SRc are similar to the ports SL, SLI, and SLc, except that an output signal SRI′ is sent to the right-neighboring tape cell.

By above, we can see that read/write and head-shift operations are correctly performed by a tape unit.

3.3.2 State module

Before introducing a state module we first explain a subroutine call mechanism. A *subroutine* is a black box having at least one calling (i.e., input) port, and at least one return (i.e., output) port (Fig. 9) that satisfies the following: If a calling signal is given to one of the calling ports, a return signal eventually comes out from one of the return ports. No signal should be given to a calling port before a return signal for the previous calling signal comes out. Here, to make a simple subroutine-call mechanism using REs, we restrict both the numbers of calling ports and return ports to be at most two. If there are two calling ports, say c^0 and c^1, we can give two kinds of information 0 and 1, regarded as an input argument, to the subroutine. Likewise, if there are two return ports r^0 and r^1, we can obtain two kinds of information 0 and 1, regarded as an output value, from the subroutine.

A tape unit acts as three subroutines by suitably specifying calling and return ports. The first one is the subroutine having the calling ports W0 and W1, and the return ports R0′ and R1′. The second has the calling port SL, and the return port SLc′. The third has the calling port SR, and the return port SRc′.

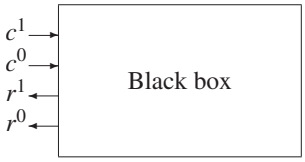


Figure 9: Subroutine. Here, it has two input ports c^0 and c^1 for calling it from a main routine, and two output ports r^0 and r^1 for returning to the main routine

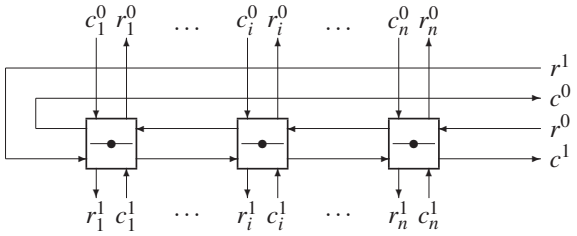


Figure 10: Subroutine caller that can be used from n points of a main routine

A *subroutine caller* is a mechanism for calling one subroutine from many points of a main routine. Fig. 10 is a caller for a subroutine having calling ports c^0 and c^1, and return ports r^0 and r^1. In this figure, c_i^s ($i \in \{1, \ldots, n\}, s \in \{0, 1\}$) is the i-th calling port for the main routine with the input s, and r_i^t ($i \in \{1, \ldots, n\}, t \in \{0, 1\}$) is the i-th return port for the main routine with the output t.

Initially, all the REs in Fig. 10 are set to the state H. If a signal is given to the port c_i^s ($i \in \{1, \ldots, n\}, s \in \{0, 1\}$), then the state of the i-th RE changes to V, and the signal goes out from the port c^s. If the signal returns via the port r^t ($t \in \{0, 1\}$), then the state of the i-th RE is restored to H, and then the signal goes out from the port r_i^t. In this way, n points of the main routine can share the same subroutine. Note that if a subroutine has only one calling port or only one return port, then unnecessary lines in the caller are removed.

A *state module* simulates one state, say q_i, of an RTM. It is shown in Fig. 11. It is composed of three submodules, which are *write-and-merge*, *head-shift*, and *read-and-branch* submodules. This figure shows the case where q_i is a right-shift state. The case for a left-shift state is similar. Because of the reversibility condition (Definition 2.2), shift direction is uniquely determined by the state. Note that a state module for an initial state consists only of a read-and-branch submodule, and that for a halting state consists of write-and-merge and head-shift submodules.

If the number of states of an RTM is m, then prepare m state modules, and connect them in a row to make a finite control of the RTM. At the left end of the array, SLc′, SRc′, R1′ and R0′ are connected to SL, SR, W1 and W0, respectively.

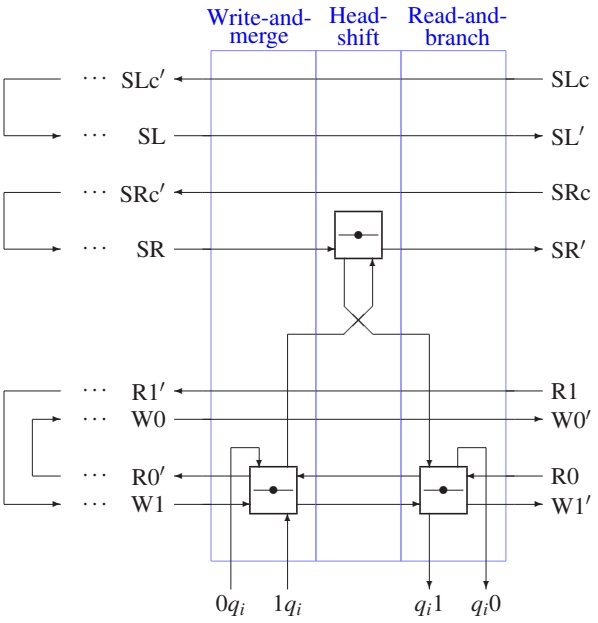


Figure 11: State module for a right-shift state q_i of an RTM

Write-and-merge submodules and read-and-branch submodules in the m state modules form a subroutine caller that share the subroutine having the calling ports W0 and W1, and the return ports R0′ and R1′ in the tape unit. By this, read/write operations on the tape unit can be performed in each state. Head-shift submodules of the right-shift states also form a subroutine caller that share the subroutine having the calling port SR, and the return port SRc′ in the tape unit. Likewise, head-shift submodules of the left-shift states form a subroutine caller that share the subroutine having the calling port SL, and the return port SLc′.

Let $T = (Q, \{0, 1\}, q_0, F, 0, \delta)$ be a two-symbol RTM. First consider how the write-and-merge submodule works. Assume $[p, s, 0, d, q_i], [p', s', 1, d, q_i] \in \delta$. Note that it also works well for the case only one of these two quintuples exists. We further assume that the write-and-merge operation is done just after a read-and-branch operation that performs a destructive readout. Thus, the tape symbol at the head position is now 0. If the submodule receives a signal from the input port $0q_i$ ($1q_i$, respectively), then it sends a calling signal to the subroutine from the port W0′ (W1′) to perform a writing operation. Since the old symbol at the head position is 0, the submodule receives a signal from the return port R0 in both cases of writing 0 and 1. By this, two different signal paths of writing 0 and 1 are reversibly merged into one, and the signal is sent to the head-shift submodule.

The head-shift submodule works as follows. If it receives a signal from the write-and-merge submodule, it sends a signal to the calling port SL′ or SR′, by which shifting is performed in the tape unit. It receives a signal from the return port SLc or SRc. Then, the signal is sent to the read-and-branch submodule.

If the read-and-merge submodule receives a signal from the head-shift submodule, it sends a signal to the calling port W0′ of the tape unit. Then the tape unit sends back a response signal via the return port R0 or R1 depending on the read symbol. The submodule finally gives a signal to the port q_i0 or q_i1. By above, read-and-branch operation is performed.

State transitions of the RTM is realized by connecting state modules in the following way. If there is a quintuple $[q_i, s, t, d, q_j] \in \delta$, then the output port q_is of the state module for q_i is connected to the input port tq_j of the state module for q_j.

3.3.3 Composing RTMs

Assembling tape cells and state modules, and connecting them as explained above, we can systematically compose a circuit made of REs that simulates any given two-symbol RTM. The circuit for the RTM T_{parity} in Example 2.1 is shown in Fig. 12. If we give a signal to the port “Start”, then it is sent to the state module for the initial state. By this, T_{parity} begins to compute. If T_{parity} halts, then the signal from the state module corresponding to the accepting or rejecting state is sent to the port “Accept” or “Reject” showing that the computation is completed.

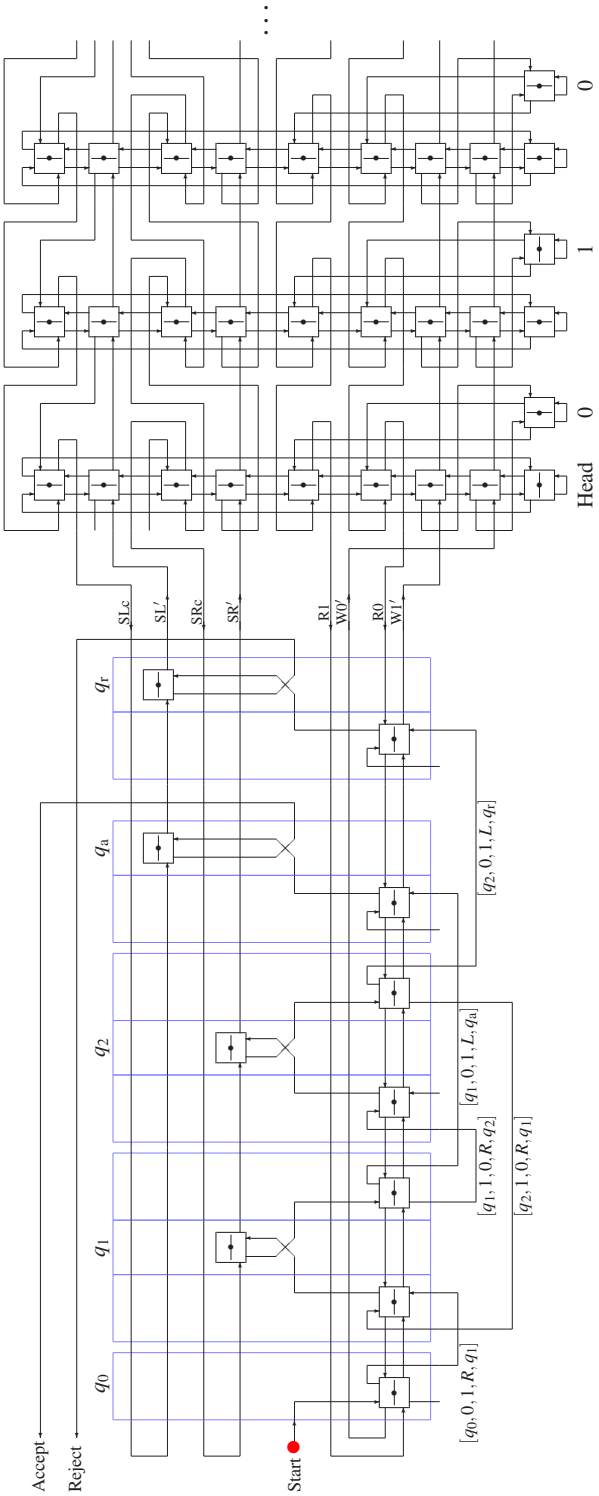


Figure 12: RTM T_{parity} composed of REs

3.4 Universality of RLEMs

There are infinitely many RLEMs even if we consider only two-state RLEMs. We use a special graphical representation for two-state RLEMs. Fig. 13 shows the representation of RLEM 3-10, where "3" means that it has three input/output symbols, and "10" is its serial number in the class of 3-symbol RLEMs. Two boxes in Fig. 13 indicate its two states. The dotted and solid lines give the input-output relation in each state. If an input signal goes through a dotted line, the state does not change (Fig. 14 **(a)**). If it goes through a solid line, the state changes (Fig. 14 **(b)**). Note that RE can be also represented by such a figure, but we employ Fig. 3 for ease in understanding.

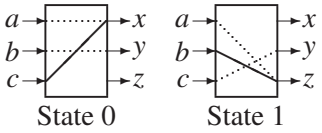


Figure 13: Two states of RLEM 3-10.

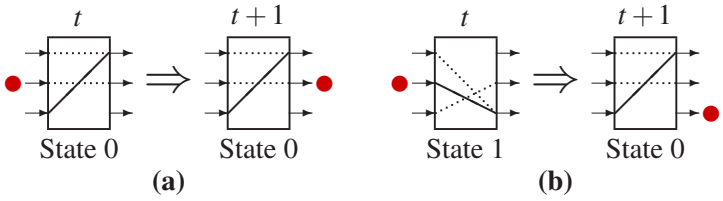


Figure 14: Operations of RLEM 3-10. **(a)** The case where the state does not change, and **(b)** the case where the state changes

Among RLEMs there are universal RLEMs in the following sense.

Definition 3.2. An RLEM R is called *universal* if any RSM can be realized by a circuit composed only of R.

As we have already seen in Sect. 3.2.2, RE is universal. We can observe that RLEM 3-10 is also universal, since RE can be composed of RLEM 3-10 as in Fig. 15 [20]. This figure shows the state H of an RE. By complementing the states of the bottom four RLEMs, we have the state V of an RE. In Fig. 15, it is easy to see that if a signal is given to the port e (w, respectively), then it goes out from the port w' (e') in two steps without changing the state of the circuit. This is a parallel case (Fig. 4 **(a)**). On the other hand, if a signal is given to the port n, then the circuit evolves as shown in Fig. 16. The signal finally goes out from the port w', and the states of the bottom four RLEMs are complemented. This is an orthogonal case (Fig. 4 **(b)**). In such a way, RE is correctly simulated.

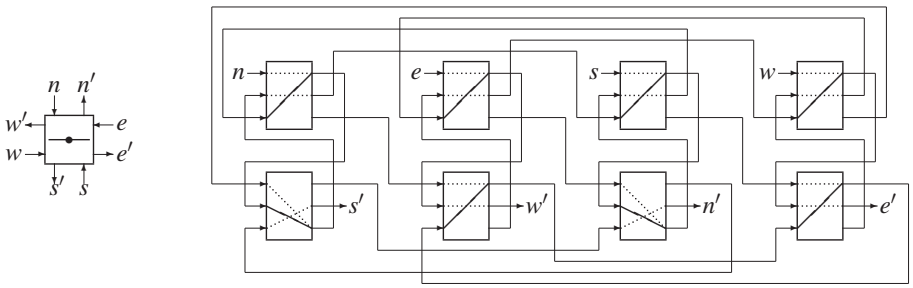


Figure 15: A circuit composed of RLEM 3-10 that simulates RE [20].

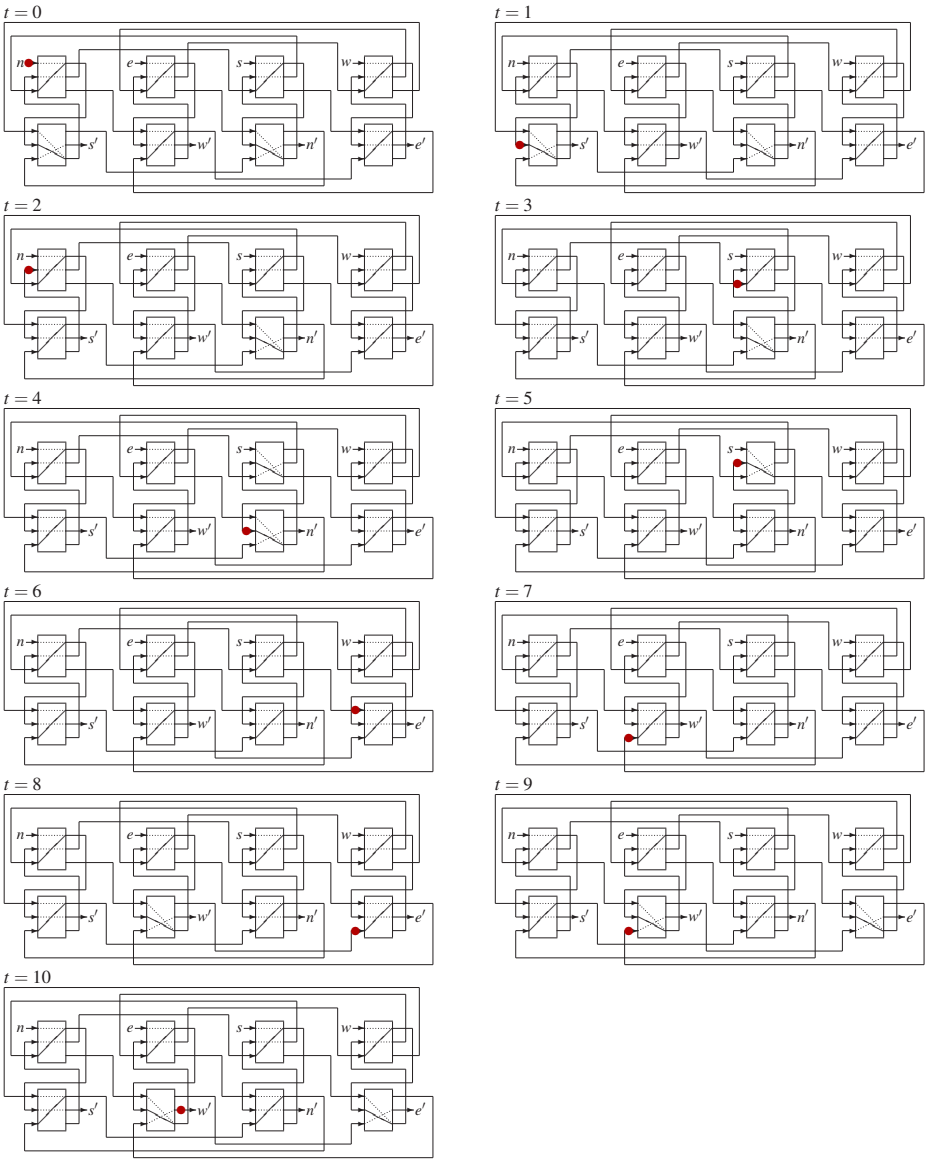


Figure 16: Process of simulating RE by RLEM 3-10 when the state of RE changes

Furthermore, we can compose RLEM 3-10 out of RLEMs 2-3 and 2-4 (Fig. 17). The circuit that simulates RLEM 3-10 is shown in Fig. 18 [10]. Hence, the set {RLEM 2-3, RLEM 2-4} is universal, though each of RLEM 2-3 and RLEM 2-4 has been proved to be non-universal [21]. This result is useful for realizing a universal RLEM such as RE in a reversible environment having a very simple microscopic law of evolution (see Sect. 4.5).

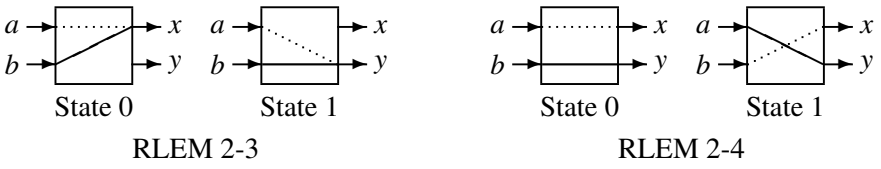


Figure 17: RLEMs 2-3 and 2-4

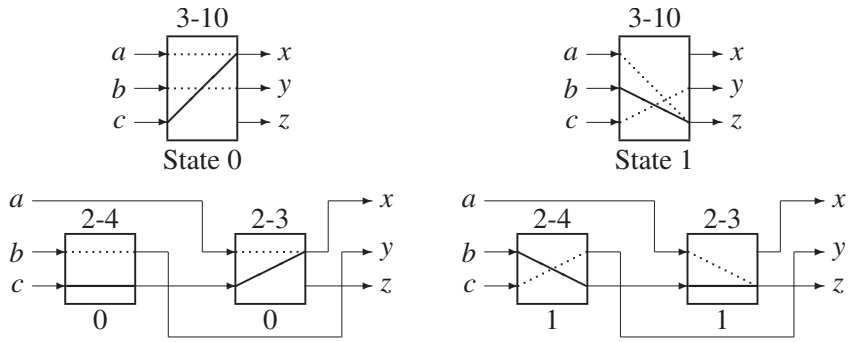


Figure 18: RLEM 3-10 is simulated by a circuit made of RLEMs 2-3 and 2-4 [10]

It is known that *every* non-degenerate two-state RLEM having three or more I/O symbols is universal. It was proved by showing the fact that, for any one of these RLEMs, there are circuits composed only of it that simulate RLEMs 2-3 and 2-4 [20]. Note that, here, a degenerate RLEM means that it is either equivalent to an RLEM with fewer I/O symbols, or equivalent to connecting wires (see [15] for its precise definition). Hence, we consider only nondegenerate RLEMs. Fig. 19 summarizes the results.

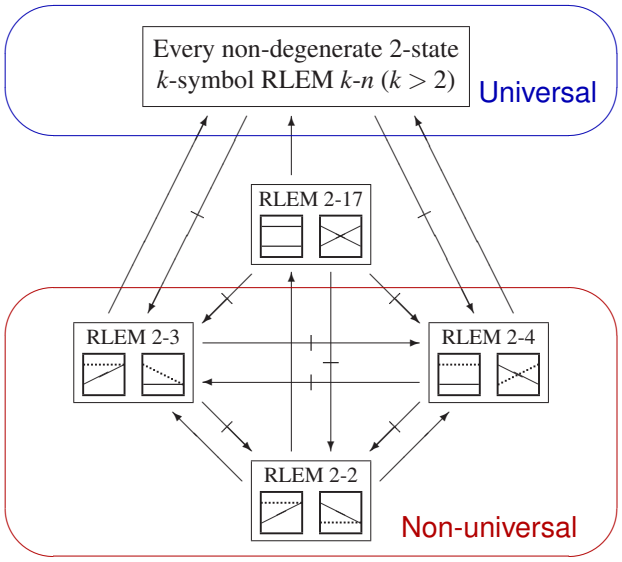


Figure 19: Universality/non-universality of two-state RLEMs [15]

4 Simple Reversible Cellular Automaton

As a reversible environment, we use a reversible cellular automaton having very simple local transition rules, which can be seen as a microscopic law of evolution.

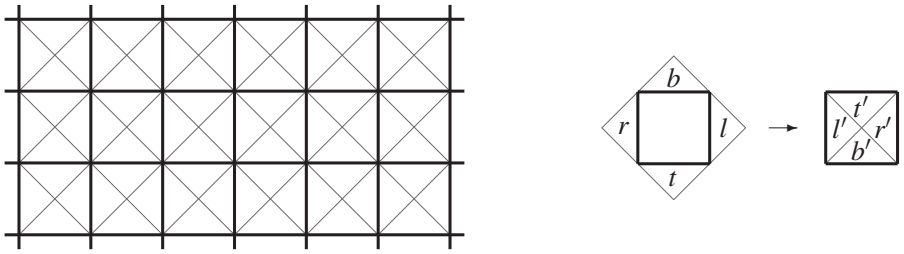
4.1 Elementary square partitioned CA (ESPCA)

A 4-neighbor *square partitioned cellular automaton* (SPCA) is a two-dimensional CA whose square cell is divided into four parts as in Fig. 20 **(a)**. The next state of a cell is determined depending on the present states of the four adjacent parts of the neighboring cells (not depending on whole the states of the four neighboring cells) as shown in Fig. 20 **(b)**. Note that the next state of a cell does not depend on the previous state of the cell itself.

Definition 4.1. A 4-neighbor *square partitioned cellular automaton* (SPCA) is defined by

$$P = (\mathbb{Z}^2, (T, R, B, L), ((0,-1), (-1,0), (0,1), (1,0)), f).$$

Here, $\mathbb{Z}^2$ is the set of all points with integer coordinates where cells are placed. The items T, R, B and L are non-empty finite sets of states of the top, right, bottom and left parts of a cell. The set of states of a cell is thus $Q = T \times R \times B \times L$. The quadruple $((0,-1), (-1,0), (0,1), (1,0))$ is a *neighborhood*. The item $f : Q \to Q$ is a *local (transition) function*.



(a) **(b)**

Figure 20: **(a)** Cellular space of a 4-neighbor square partitioned cellular automaton (SPCA), and **(b)** its local transition rule $f(t, r, b, l) = (t', r', b', l')$

If $f(t, r, b, l) = (t', r', b', l')$ holds for $(t, r, b, l), (t', r', b', l') \in Q$, this relation is called a *local transition rule* of P. It is also indicated as in Fig. 20 **(b)**. The local function f is thus defined by a set of local transition rules.

Definition 4.2. Let $P = (\mathbb{Z}^2, (T, R, B, L), ((0, -1), (-1, 0), (0, 1), (1, 0)), f)$ be an SPCA. A *configuration* of P is a function $\alpha : \mathbb{Z}^2 \to Q$. The set of all configurations of P is denoted by $\mathrm{Conf}(P)$, i.e., $\mathrm{Conf}(P) = \{\alpha \mid \alpha : \mathbb{Z}^2 \to Q\}$. Let $\mathrm{pr}_T : Q \to T$ be the *projection function* that satisfies $\mathrm{pr}_T(t, r, b, l) = t$ for all $(t, r, b, l) \in Q$. The projection functions $\mathrm{pr}_R : Q \to R$, $\mathrm{pr}_B : Q \to B$ and $\mathrm{pr}_L : Q \to L$ are defined similarly. The *global function* $F : \mathrm{Conf}(P) \to \mathrm{Conf}(P)$ of P is defined as the one that satisfies the following.

$$\begin{aligned} &\forall \alpha \in \mathrm{Conf}(P), \forall (x, y) \in \mathbb{Z}^2 : \\ &\quad F(\alpha)(x, y) \;=\; f(\mathrm{pr}_T(\alpha(x, y-1)), \mathrm{pr}_R(\alpha(x-1, y)), \mathrm{pr}_B(\alpha(x, y+1)), \\ &\qquad\qquad \mathrm{pr}_L(\alpha(x+1, y))) \end{aligned}$$

Definition 4.3. An SPCA P is called *reversible* if its global function is injective.

As for the notions related to reversibility, see Sect. 10.3 of [15] for the details. The next Lemma shows that injectivity of the global function of a PCA is equivalent to that of the local function [15, 19]. By this, we can easily obtain a reversible CA, since it is sufficient to design a PCA whose local function is injective.

Lemma 4.1. *Let P be an SPCA. Its global function F is injective if and only if its local function f is injective.*

Here, we define the simplest subclass of SPCAs such that its local function is rotation-symmetric, and each of four parts has only two states. It is called an *elementary* SPCA (ESPCA) as in the case of a one-dimensional *elementary cellular automaton* (ECA) [28]. We first define the notion of rotation-symmetry.

Definition 4.4. Let $P = (\mathbb{Z}^2, (T, R, B, L), ((0,-1), (-1,0), (0,1), (1,0)), f)$ be an SPCA. The SPCA P is called *rotation-symmetric* (or *isotropic*) if the following conditions (1) and (2) hold.

(1) $T = R = B = L$

(2) $\forall (t, r, b, l), (t', r', b', l') \in T \times R \times B \times L$:
$f(t, r, b, l) = (t', r', b', l') \Rightarrow f(r, b, l, t) = (r', b', l', t')$

Definition 4.5. Let $P = (\mathbb{Z}^2, (T, R, B, L), ((0,-1), (-1,0), (0,1), (1,0)), f)$ be an SPCA. We say P is an *elementary triangular partitioned cellular automaton* (ESPCA), if $T = R = B = L = \{0, 1\}$, and it is rotation-symmetric.

Since an ESPCA is rotation-symmetric, its local function $f : \{0,1\}^4 \to \{0,1\}^4$ is defined by only six local transition rules, which are described by the following six values.

$$f(0,0,0,0),\ f(0,0,1,0),\ f(0,0,1,1),\ f(1,0,1,0),\ f(0,1,1,1),\ f(1,1,1,1)$$

Here, $f(0,0,1,0),\ f(0,0,1,1),\ f(0,1,1,1) \in \{0,1\}^4$. However, $f(1,0,1,0) \in \{(0,0,0,0),\ (0,1,0,1),\ (1,0,1,0),\ (1,1,1,1)\}$ and $f(0,0,0,0), f(1,1,1,1) \in \{(0,0,0,0), (1,1,1,1)\}$, since it is rotation-symmetric. Hence, there are $16^3 \times 4 \times 2^2 = 65,536$ ESPCAs in total.

Reading the 4-bit values of $f(0,0,0,0),\ f(0,0,1,0),\ f(0,0,1,1),\ f(1,0,1,0),\ f(0,1,1,1),\ f(1,1,1,1)$ as six binary numbers, we can express an ESPCA by a 6-digit hexadecimal identification number $uvwxyz$. For example, if $f(0,0,1,0) = (t, r, b, l)$, then $v = 2^3 t + 2^2 r + 2^1 b + 2^0 l$. An ESPCA with the identification number $uvwxyz$ is denoted by ESPCA $uvwxyz$.

4.2 A particular ESPCA P_0

Here, we consider a particular reversible ESPCA with the identification number 01caef. Hereafter, it is denoted by P_0 for short. It is first studied in [17]. Fig. 21 shows a pictorial representation of the six local transition rules of ESPCA 01caef. Though its local function is simple, its behavior is complex. Therefore, it is generally difficult to follow evolution processes of P_0 using only paper and pencil. To see its evolution processes, we created an emulator of P_0 on the general purpose CA simulator *Golly* [27]. The emulator files and pattern files for P_0 are available in [16].

It is easy to see that the local function of P_0 is injective. Therefore it is a reversible ESPCA. An ESPCA is called *conservative* if the number of particles (i.e., state 1) is conserved in each local transition rule. It is an analog of various conservation laws in physics. We can see that ESPCA P_0 is conservative.

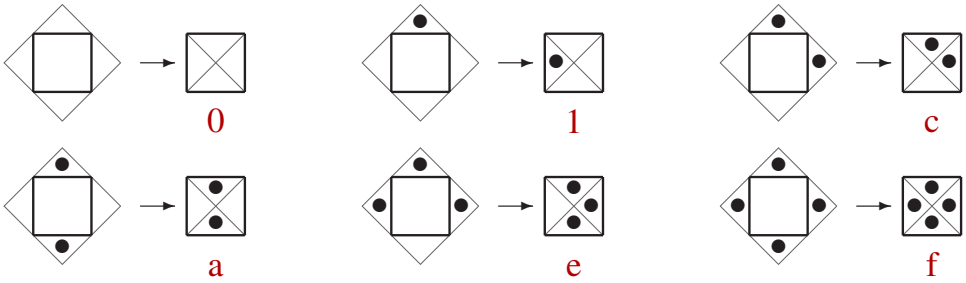


Figure 21: Local function defined by the six local transition rules of a particular reversible and conservative ESPCA 01caef, which is denoted by P_0 hereafter

4.3 Useful patterns in ESPCA P_0

A *pattern* is a finite segment of a configuration (see Sect. 10.2.1 of [15] for its precise definition). A *periodic pattern* is one such that the same pattern appears at the same position after some time steps. The periodic pattern given in Fig. 22 is called a *blinker*. It is of period 2. Though there are many kinds of periodic patterns in P_0, a blinker is particularly useful among them as we shall see in Sect. 4.4

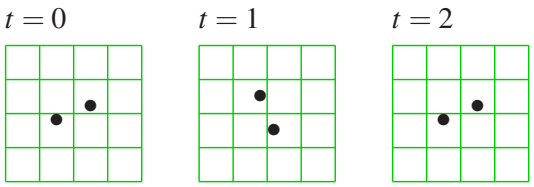


Figure 22: Blinker in P_0 [17]. It is of period 2

A *space-moving pattern* is one such that the same pattern appears at a different position after some time steps. In P_0 there are many kinds of space-moving patterns of various periods [17]. In fact, if we start from a random-like pattern, then we often observe that space-moving patterns appear.

The pattern having the shortest period among the space-moving patterns so far found is called a *glider* (Fig. 23). It travels one cell diagonally in 12 steps. It will be used as a signal when constructing reversible Turing machines, since it has interesting and desirable properties as described in Sect. 4.4.

The pattern shown in Fig. 24 is called a *block*. It is a *stable pattern*, which is a periodic pattern of period 1, and thus does not change its pattern if no other pattern touches it. In the following, it will be used only for writing comments and indicating a border of a logic element in the cellular space. Hence, it has no functional role for composing reversible Turing machines.

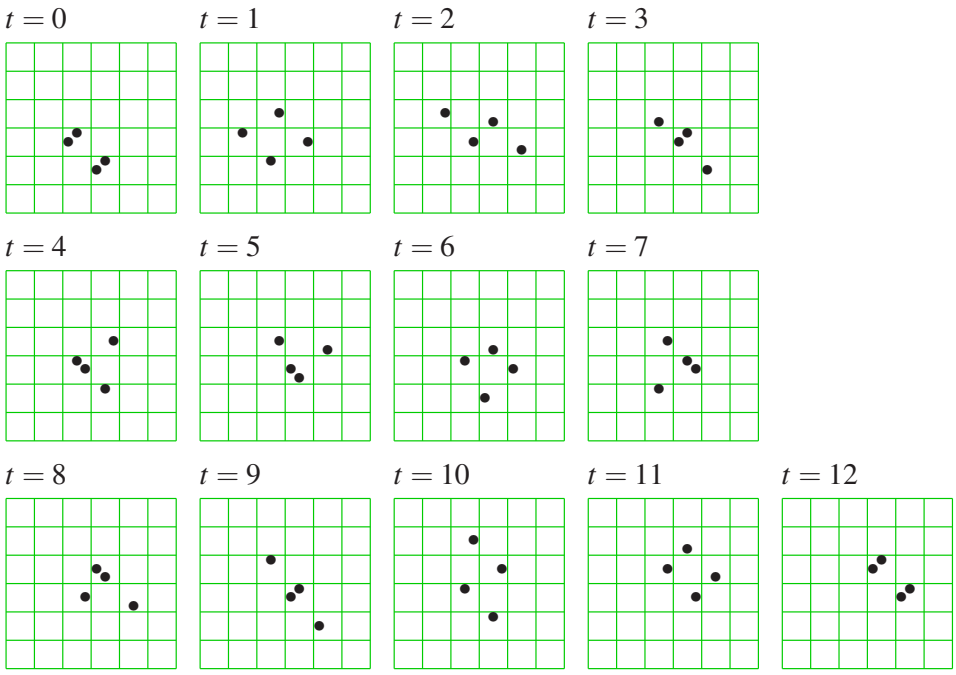


Figure 23: Glider in P_0 [17]. It is of period 12

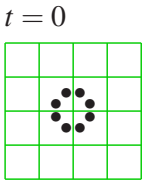


Figure 24: Block in P_0 [17]. It is a stable pattern

4.4 Three useful phenomena in ESPCA P_0

We make three experiments of interacting a glider with a blinker. However, each of these experiment needs a large number of steps. In particular the third experiment takes more than 2000 steps. Therefore, it is not possible to show correctness of the results in this paper. They are verified by computer simulation [16].

The first experiment is shown in Fig. 25. Colliding a glider with a blinker in this manner, a right-turn of a glider is realized.

The second experiment is in Fig. 26. By this, a glider makes a U-turn. It is used to test if a blinker exists or not at a specified position. It is also used to reversibly merge two signal paths into one (it is explained in Sect. 4.5).

The third experiment is in Fig. 27. By this, the position of the blinker is shifted by 6 cells, and the glider makes a right-turn. Using this phenomenon, a kind of

memory device is realized, where the memory states are kept by the positions of the blinker. At the same time, it can test if a blinker exists at a specified position, and can merge two signal paths into one.

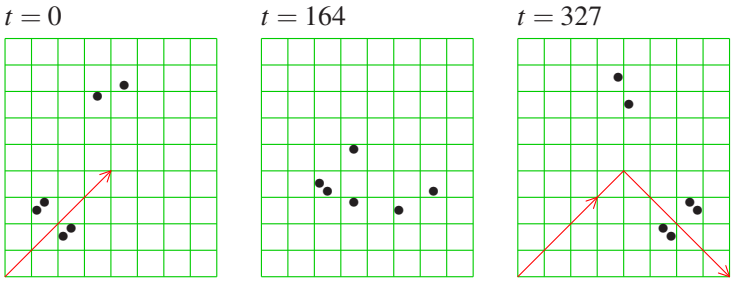


Figure 25: Right-turn of a glider in P_0 [17]

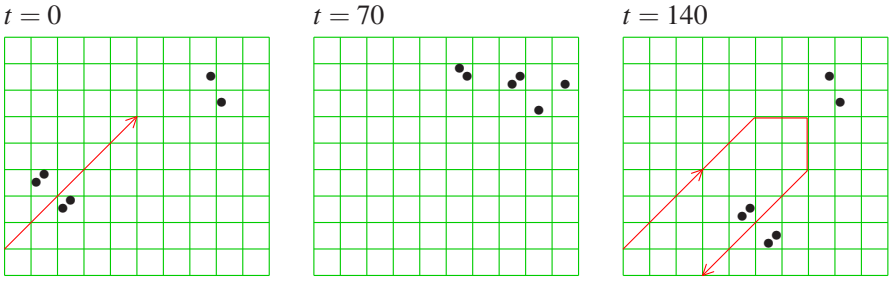


Figure 26: U-turn of a glider in P_0 [17]

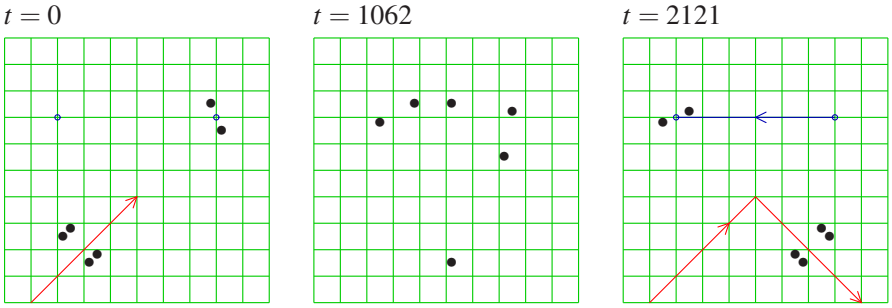


Figure 27: Shifting a blinker by a glider in P_0 [17]

4.5 Composing RLEMs in ESPCA P_0

Using three useful phenomena given in Sect. 4.4, we implement RLEMs in P_0. Since it is difficult to make RE directly, we first compose RLEMs 2-3 and 2-4. Using them, we make RLEM 3-10, and then RE. In [17], RLEM 4-31 is implemented in P_0, whose pattern size is smaller than that of RE. However, here, we use RE, since its operation is easier to understand. Note that the essential parts of RLEMs constructed here consist only of blinkers, since blocks are used to write comments and to indicate borders of the RLEMs.

RLEM 2-3

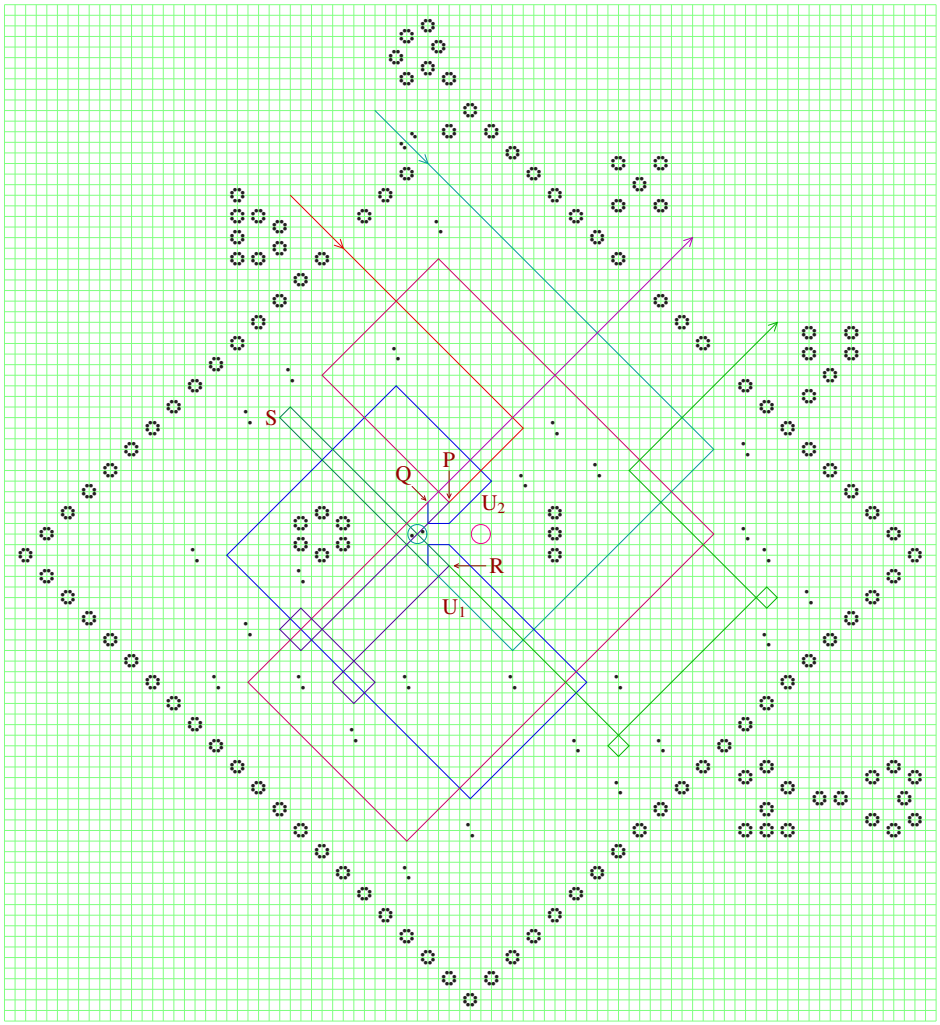


Figure 28: RLEM 2-3 implemented in P_0

The pattern shown in Fig. 28 simulates RLEM 2-3. There are many blinkers in this pattern. One is used as a *position marker* for keeping the memory state 0 or 1, while others are used for turning a signal. Two small circles near the center of the pattern show possible positions of the position marker. If the marker is at the left (right, respectively) position, we regard that the RLEM is in the state 0 (1).

First, consider the case where the state is 0 and an input signal is given to the port a as in this figure. The signal makes a U-turn at the U-turn gadget U_1 since the state is 0. Then it goes to the gadget U_2, and again makes a U-turn passing through Q. Note that U_2 is used to reversibly merge the path with that of the second case. Finally the signal goes out from the port x.

Second, consider the case where the state is 0 and an input signal is given to the port b. At P the signal shifts the position marker to the right, and makes a right-turn. Thus, the state changes to 1. Then, the signal goes out from the output port x via the point Q. This signal path is merged with that of the first case at Q.

Third, consider the case where the state is 1 and an input signal is given to the port a. In this case, the signal goes out from the output port y via S and R without interacting the position marker.

Fourth, consider the case where the state is 1 and an input signal is given to the port b. The signal goes straight ahead at the point P. Then, it shifts the position marker to the left and makes a right-turn at R. Thus, the state changes to 0. Finally it goes out from y. This signal path is merged with that of the third case at R.

Note that, in an RLEM, an incoming signal interacts with the state of the RLEM, not with other signals. Therefore, there is no need of synchronizing two or more signals as in the case of logic gates. Therefore, it greatly simplifies implementation of RLEMs and connecting them in P_0

RLEM 2-4

The pattern shown in Fig. 29 simulates RLEM 2-4. As in the case of RLEM 2-3, one blinker near the center of the pattern is used as a position marker for keeping the memory state 0 or 1. If the marker is in the right (left, respectively) small circle, we regard that the RLEM is in the state 0 (1).

First, consider the case where the state is 0 and an input signal is given to the port a. The signal makes a U-turn at U_2. Then it goes to U_1, and again makes a U-turn passing through T. Finally the signal goes out from the port x.

Second, consider the case where the state is 1 and an input signal is given to the port b. At R the signal goes straight ahead. Then it passes through the points S and T. Finally it goes out from the port x. This signal path is merged with that of the first case at T.

Third, consider the case where the state is 1 and an input signal is given to the port a. The signal goes straight ahead at Q. Then, at P it shifts the position marker

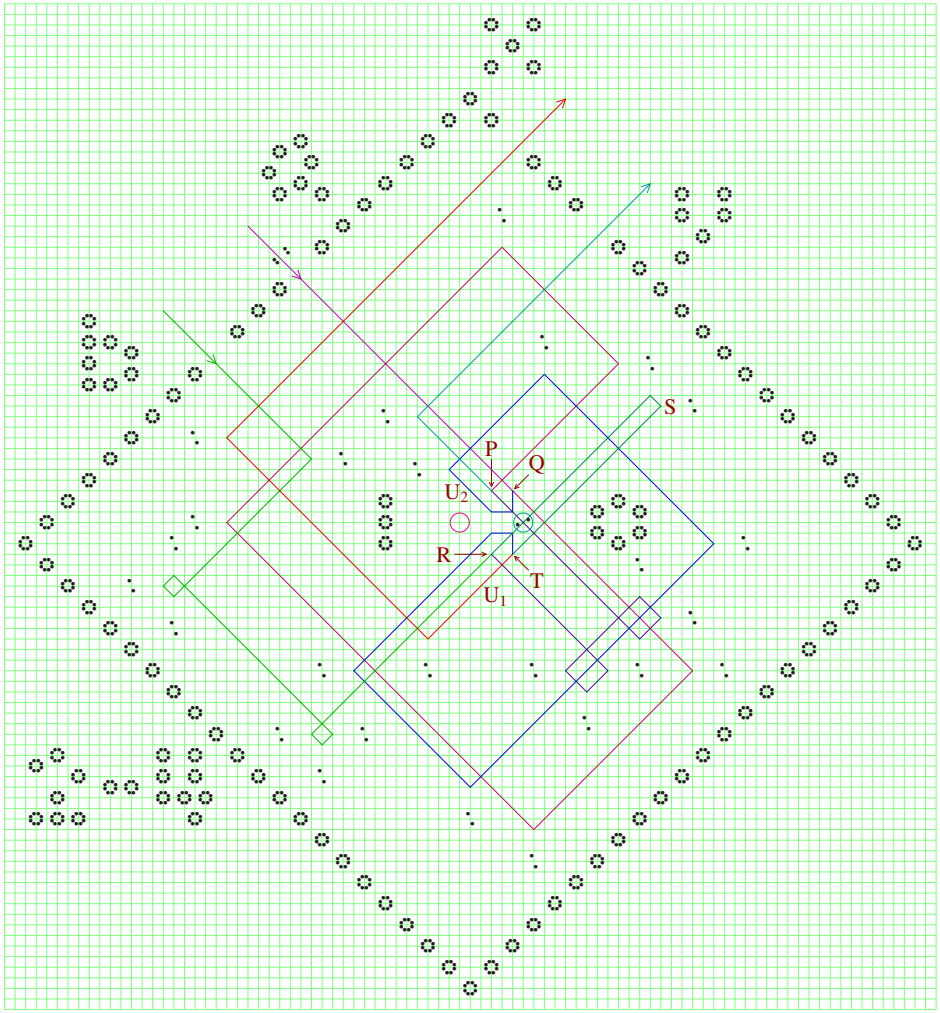


Figure 29: RLEM 2-4 implemented in P_0

to the right, and makes a right-turn. By this, the state changes to 0. Finally it goes out from the port y.

Fourth, consider the case where the state is 0 and an input signal is given to the port b. At R the signal shifts the position marker to the left, and makes a right-turn. By this, state changes to 1. Then, it passes through P, and finally goes out from y. In this case, the signal path is merged with that of the third case at P.

It should be noted that the move function of RLEM 2-4 is the inverse of that of RLEM 2-3. The move function of RLEM 2-3 is as follows.

$$(0,a) \mapsto (0,x),\ (0,b) \mapsto (1,x),\ (1,a) \mapsto (1,y),\ (1,b) \mapsto (0,y)$$

Its inverse is as follows, and is isomorphic to that of RLEM 2-4.

$$(0,x) \mapsto (0,a),\ (1,x) \mapsto (0,b),\ (1,y) \mapsto (1,a),\ (0,y) \mapsto (1,b)$$

In [18], it is shown that in a reversible triangular partitioned CA (ETPCA), a pattern for the "inverse functional module" can be easily obtained from the original pattern by a simple transformation. This is due to the time-symmetry [18] of reversible ETPCAs. A similar property also holds for reversible ESPCAs (but its details are omitted here). By this, the pattern in Fig. 29 is obtained by putting blinkers at the mirror image positions of blinkers of the pattern of in Fig. 28.

RLEM 3-10

Combining the patterns for RLEMs 2-3 and 2-4 to form the circuit shown in Fig. 18, we can easily obtain a pattern that simulates RLEM 3-10 as in Fig. 30.

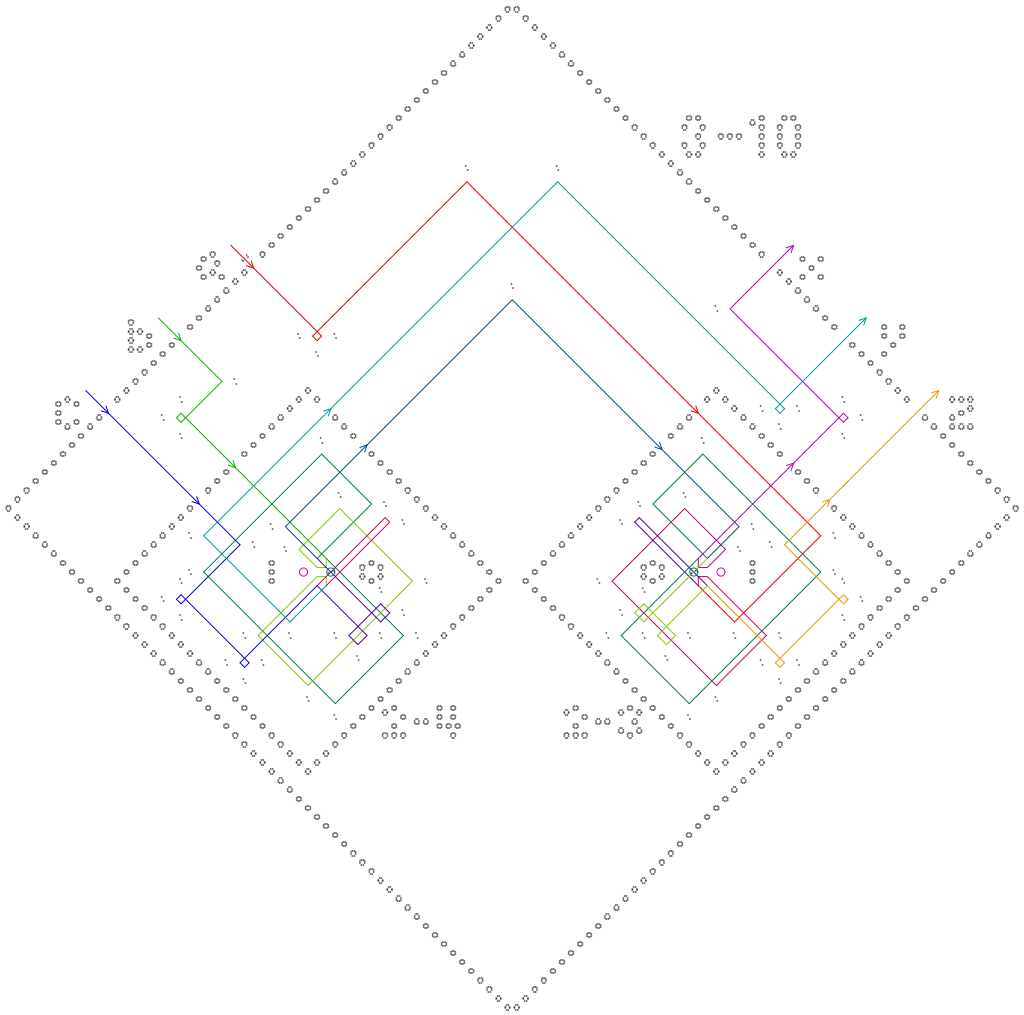


Figure 30: RLEM 3-10 implemented in P_0 composed of RLEMs 2-3 and 2-4

RE

Placing eight copies of the pattern of RLEM 3-10 (Fig. 30) and connecting them to form the circuit shown in Fig. 15, we can obtain a pattern for RE. However, since it is very large (2,000 × 2,000), its figure is omitted here (the pattern can be seen using the file `08_RE_by_3-10.rle` in [16] on Golly).

4.6 Making RTMs in ESPCA P_0

Putting copies of the patterns of RE in P_0 at the positions of REs in Fig. 12, and connecting them appropriately, we have a configuration of P_0 that simulates the RTM T_{parity} of Fig. 12. Any RTM can be implemented in P_0 in this manner.

Fig. 31 shows the configuration for the RTM T_{power} in Example 2.2 simulated on Golly [16]. It takes more than one billion (1,137,250,105) steps to have an answer for the unary input $n = 4$. Therefore, when simulating the computing process of T_{power} on Golly, its speeding-up mode should be used.

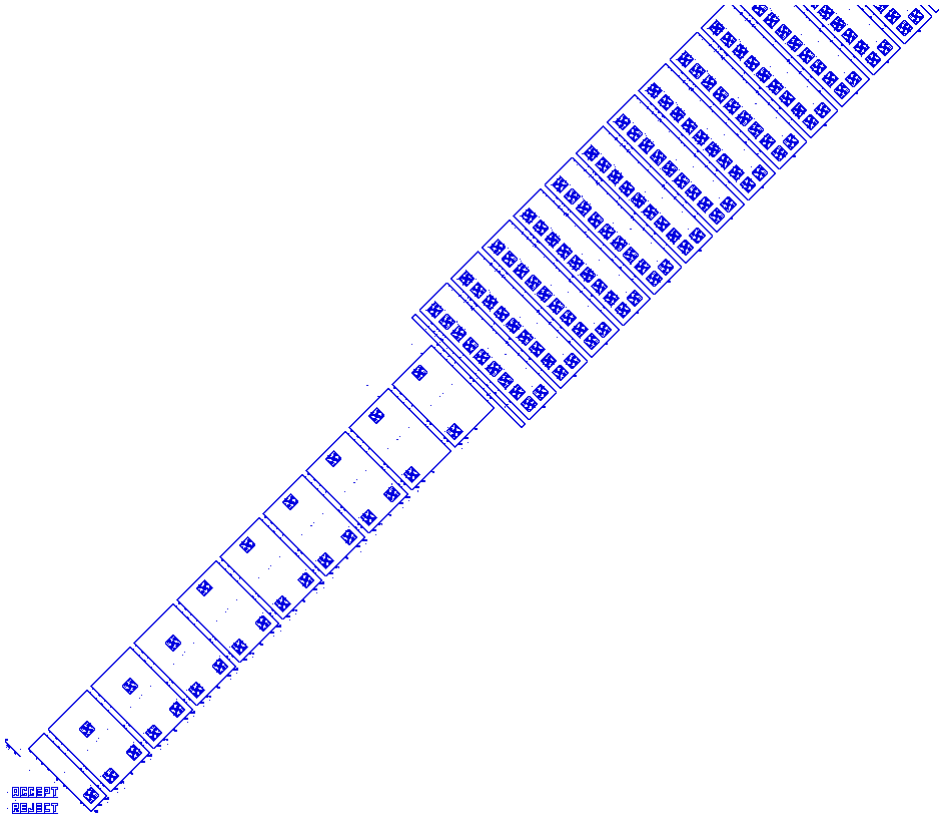


Figure 31: RTM T_{power} implemented in ESPCA P_0 simulated on Golly [16]. Each small square is a pattern that simulates RE

5 Concluding Remarks

We saw that even from a very simple reversible microscopic law, reversible computing machines like RTMs can be realized by a construction method shown in Fig. 1. Here we used a particular reversible ESPCA P_0 as a reversible environment. However, there are many possibilities of other simple reversible CAs, or other frameworks of reversible environments. For example, in [18], a reversible elementary triangular partitioned cellular automaton (ETPCA) is used to construct RTMs. An important fact observed in this paper is that only a few useful reversible phenomena (shown in Sect. 4.4) are sufficient to compose reversible computing machines. We expect such a fact also holds in various reversible environments.

Acknowledgements
The author is grateful to the reviewers for their valuable comments. He also express his thanks to the development team of Golly [27].

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] Cook, M.: Universality in elementary cellular automata. Complex Syst. **15**, 1–40 (2004)

[3] Fredkin, E., Toffoli, T.: Conservative logic. Int. J. Theoret. Phys. **21**, 219–253 (1982). https://doi.org/10.1007/BF01857727

[4] Gurevich, Y.: Reversify any sequential algorithm. Bulletin of EATCS **134**, 42–65 (2021)

[5] Hartmanis, J., Stearns, R.: On the computational complexity of algorithms. Trans. Amer. Math. Soc. **117**, 285–306 (1965). https://doi.org/10.2307/1994208

[6] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Prentice Hall (2006)

[7] Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: Proc. 36th FOCS. pp. 66–75. IEEE (1997). https://doi.org/10.1109/SFCS.1997.646094

[8] Landauer, R.: Irreversibility and heat generation in the computing process. IBM J. Res. Dev. **5**, 183–191 (1961). https://doi.org/10.1147/rd.53.0183

[9] Lecerf, Y.: Machines de Turing réversibles — récursive insolubilité en $n \in \mathbf{N}$ de l'équation $u = \theta^n u$, où θ est un isomorphisme de codes. Comptes Rendus Hebdomadaires des Séances de L'académie des Sciences **257**, 2597–2600 (1963)

[10] Lee, J., Peper, F., Adachi, S., Morita, K.: An asynchronous cellular automaton implementing 2-state 2-input 2-output reversed-twin reversible elements. In: Proc. ACRI 2008 (eds. H. Umeo, et al.), LNCS 5191. pp. 67–76 (2008). https://doi.org/10.1007/978-3-540-79992-4_9

[11] Margolus, N.: Physics-like model of computation. Physica D **10**, 81–95 (1984). https://doi.org/10.1016/0167-2789(84)90252-5

[12] Morita, K.: Universality of a reversible two-counter machine. Theoret. Comput. Sci. **168**, 303–320 (1996). https://doi.org/10.1016/S0304-3975(96)00081-3

[13] Morita, K.: A simple reversible logic element and cellular automata for reversible computing. In: Proc. MCU 2001 (eds. M. Margenstern, Y. Rogozhin), LNCS 2055. pp. 102–113 (2001). https://doi.org/10.1007/3-540-45132-3_6

[14] Morita, K.: A deterministic two-way multi-head finite automaton can be converted into a reversible one with the same number of heads. In: Proc. RC 2012 (eds. R. Glück, T. Yokoyama), LNCS 7581. pp. 29–43 (2013). https://doi.org/10.1007/978-3-642-36315-3_3

[15] Morita, K.: Theory of Reversible Computing. Springer, Tokyo (2017). https://doi.org/10.1007/978-4-431-56606-9

[16] Morita, K.: Data set for simulating a reversible elementary square partitioned cellular automaton with the ID number 01caef on Golly. Hiroshima University Institutional Repository, http://ir.lib.hiroshima-u.ac.jp/00051974 (2021)

[17] Morita, K.: Computing in a simple reversible and conservative cellular automaton. In: Proc. First Asian Symposium on Cellular Automata Technology (eds. S. Das, G.J. Martinez) AISC 1425, Springer pp. 3–16 (2022). https://doi.org/10.1007/978-981-19-0542-1_1

[18] Morita, K.: Gliders in the Game of Life and in a reversible cellular automaton. In: The Mathematical Artist – A Tribute To John Horton Conway (eds. S. Das, S. Roy, K. Bhattacharjee). Springer (in press)

[19] 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

[20] 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 and Soft Computing **18**, 37–54 (2012)

[21] Mukai, Y., Ogiro, T., Morita, K.: Universality problems on reversible logic elements with 1-bit memory. Int. J. Unconventional Computing **10**, 353–373 (2014)

[22] Neary, T., Woods, D.: Four small universal Turing machines. Fundam. Inform. **91**, 123–144 (2009). https://doi.org/10.3233/FI-2009-0036

[23] Rogozhin, Y.: Small universal Turing machines. Theoret. Comput. Sci. **168**, 215–240 (1996). https://doi.org/10.1016/S0304-3975(96)00077-1

[24] Shannon, C.E.: A universal Turing machine with two internal states. In: Automata Studies. pp. 157–165. Princeton University Press, Princeton, NJ (1956)

[25] Toffoli, T.: Computation and construction universality of reversible cellular automata. J. Comput. Syst. Sci. **15**, 213–231 (1977). https://doi.org/10.1016/S0022-0000(77)80007-X

[26] Toffoli, T.: Reversible computing. In: Automata, Languages and Programming (eds. J.W. de Bakker, J. van Leeuwen), LNCS 85. pp. 632–644 (1980). https://doi.org/10.1007/3-540-10003-2_104

[27] 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. http://golly.sourceforge.net/ (2005)

[28] Wolfram, S.: A New Kind of Science. Wolfram Media Inc. (2002)