# Reversible computing and cellular automata — a survey [*]

(Preliminary draft)

Kenichi Morita

Hiroshima University, Higashi-Hiroshima 739-8527, Japan

morita@iec.hiroshima-u.ac.jp

**Abstract**

Reversible computing is a paradigm where computing models are so defined that they reflect physical reversibility, one of the fundamental microscopic physical property of Nature. In this survey/tutorial paper, we discuss how computation can be carried out in a reversible system, how a universal reversible computer can be constructed by reversible logic elements, and how such logic elements are related to reversible physical phenomena. We shall see that, in reversible systems, computation can be often carried out very different manner from conventional (i.e., irreversible) computing systems, and even very simple reversible systems or logic elements have computation- or logical-universality. We discuss these problems based on reversible logic elements/circuits, reversible Turing machines, reversible cellular automata, and some other related models of reversible computing.

*Key words:* reversible logic element, reversible Turing machine, reversible cellular automata, computation-universality

## 1 Introduction

Physical reversibility is one of the fundamental microscopic laws of Nature. Since future computing devices will surely be implemented based on some physical phenomena of nano-scale level, it is an important problem how such property of Nature can be effectively used in computing. The definition of reversible computing systems is rather simple. They are the ones such that each of their computational configurations has at most one previous configuration. Hence, every computation process can be traced backward uniquely from the end to the start. In other words, they are backward deterministic systems. Though the definitions of them is thus rather simple, it is remarkable that they reflect physical reversibility very well.

[*]Supported by Grant-in-Aid for Scientific Research (C) No. 16500012 from JSPS.

Landauer [19] first argued the relation between logical reversibility and physical reversibility. He posed "Landauer's principle" stating that an irreversible logical operation, such as erasure of an unnecessary information, inevitably causes heat generation. Bennett [4] studied reversible Turing machines from this viewpoint, and showed that for any irreversible Turing machine we can construct a reversible one that simulates the former and leaves no garbage information on its tape when it halts. He also pointed out a possibility of physically reversible computer where dissipation of energy is arbitrarily small.

Since then, several reversible computing models, such as reversible cellular automata, and reversible logic elements and circuits, have been proposed and investigated from the above viewpoint. Toffoli [47] showed that every irreversible $k$-dimensional cellular automaton can be simulated by some $k + 1$-dimensional reversible one, hence two-dimensional reversible cellular automata are computation-universal. Later, Morita and Harao [30] proved computation-universality of one-dimensional reversible cellular automata. As for reversible logic elements and circuits, Toffoli [48, 49] first studied them in the relation to physical reversibility. Then, Fredkin and Toffoli [11] introduced "conservative logic," and showed that any logic function can be realized by a garbageless circuit composed of the Fredkin gate, a reversible logic gate. They also proposed an interesting physical model of reversible computing called the Billiard Ball Model (BBM), and showed that any reversible logic circuit composed of Fredkin gates can be realized in BBM. After these earlier works, various simple reversible models having computation-universality, as well as other models of reversible computing, have been proposed and investigated.

In this survey/tutorial paper, we discuss how computation is performed in reversible computing systems, and how different they are from conventional (i.e., irreversible) computing systems. We shall see that even very simple reversible systems have computation-universality, and there are also simple reversible primitives from which reversible computers can be built. We can see, in these systems, computation is often carried out in a very unique manner not used in conventional computing systems, thus they may give new ways and concepts for future computing.

An outline of this paper is as follows. In Section 2, reversible logic elements and circuits, and their realization in Billiard Ball Model are discussed. Section 3 deals with reversible Turing machines (RTMs), their realization by reversible logic elements, and small universal RTMs. In Section 4, reversible cellular automata (RCAs) are discussed. In particular, we see how computation can be performed in simple RCAs. Section 5 gives remarks and future directions.

## 2 Reversible logic elements and circuits

The concept of reversibility can be defined for several different levels of computing: from an element level (i.e., a level of logic elements, or even a physical realization level) to a system level (i.e., a level of computing systems, or a program level). We shall see later that the notions of reversibility in these different levels are closely related to each other. For example, reversible Turing machines can be composed of reversible logic elements, and the latter are further realized in a reversible physical model. In this section we deal with reversible logic elements and circuits. There are two types of logic elements: one without memory, which is usually called a logic gate, and one with memory. As we shall see, the latter one (i.e., an element with memory) is often useful in reversible computing. Here, we also discuss the Billiard Ball Model, which is a reversible physical model of computing introduced by Fredkin and Toffoli [11], and realization methods of reversible logic elements in it.

### 2.1 Reversible logic gates

Computers of nowadays are composed of logic gates and memories. Among typical traditional logic gates, AND, OR, and NAND are "irreversible" in the sense that we cannot retrieve their input only from the output, because these gates realize logic functions that are not one-to-one. On the other hand, NOT is reversible since it realizes a one-to-one function. A reversible logic gate is a one that realizes a one-to-one logic function, hence it is, in general, a many-input many-output gate. Early study on such kinds of gates was made by Petri [44]. Later, Toffoli [48, 49], and Fredkin and Toffoli [11] studied them in connection with physical reversibility.

#### 2.1.1 The Fredkin gate and the Toffoli gate

An $m$-input $n$-output logic gate that realizes a logic (i.e., Boolean) function $f : \{0,1\}^m \to \{0,1\}^n$ $(m \le n)$ is called *reversible*, if $f$ is one-to-one. Though there are many reversible logic gates, we discuss here a few typical ones, especially the Fredkin gate and the Toffoli gate.

The *Fredkin gate* is a 3-input 3-output logic gate that realizes the following function (Fig. 1).

$$f_{\mathrm{F}} : (c, p, q) \mapsto (c, c \cdot p + \bar{c} \cdot q, \bar{c} \cdot p + c \cdot q)$$

It is easy to see $f_{\mathrm{F}}$ is one-to-one. Furthermore, we can see $f_{\mathrm{F}}^{-1} = f_{\mathrm{F}}$.

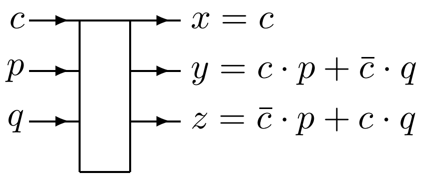

$c$
$p$
$q$
$x = c$
$y = c \cdot p + \bar{c} \cdot q$
$z = \bar{c} \cdot p + c \cdot q$

Figure 1: The Fredkin gate.

The *generalized AND/NAND gate* was proposed by Toffoli [48, 49]. It realizes the following logic function.

$$\theta^{(n)} : (x_1, \cdots, x_{n-1}, x_n) \mapsto (x_1, \cdots, x_{n-1}, (x_1 \cdots x_{n-1}) \oplus x_n)$$

It is also easy to see that it is a reversible logic gate, and $(\theta^{(n)})^{-1} = \theta^{(n)}$. The gate that realizes $\theta^{(2)}$ is sometimes called the *controlled NOT (CNOT)*, and the gate for $\theta^{(3)}$ is called the *Toffoli gate* (Fig. 2).

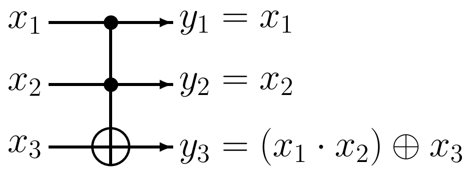


Figure 2: The generalized AND/NAND gate of order 3 called the Toffoli gate.

Besides reversibility, the Fredkin gate has bit-conserving property: the total number of 1's is conserved between the input lines and the output lines. But, the Toffoli gate is not so. This property is an analogue of the conservation law of mass, energy, or momentum in physics. Fredkin and Toffoli [11] proposed a design theory of reversible logic circuits called "conservative logic," where the Fredkin gate is used as a logical primitive. There, fan-out (i.e., branching) of an output is not allowed, because it violates the conservation law (if it is implemented in a microscopic level). Hence, if fan-out is needed, we should construct a circuit that realizes fan-out by using Fredkin gates.

A set of logic elements $E$ is called *logically universal*, if any logic function can be realized by a circuit using only the elements in $E$. As shown in Fig. 3, AND, NOT, and fan-out can be realized by Fredkin gates [11], if we allow to supply constant inputs 0's and 1's, and allow to generate garbage (useless) outputs besides the true inputs and outputs. Since any logic function can be composed of AND, NOT, and fan-out, logical universality of the set {Fredkin gate} is concluded. Likewise, AND, NOT, and fan-out can be realized by Toffoli gates as in Fig. 4.

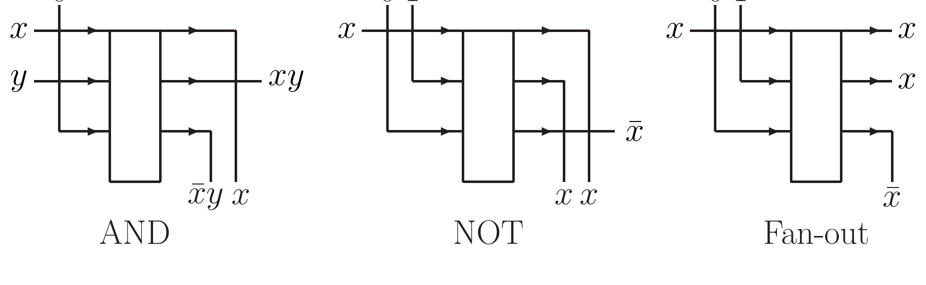


Figure 3: Implementing AND, NOT, and fan-out by Fredkin gates.

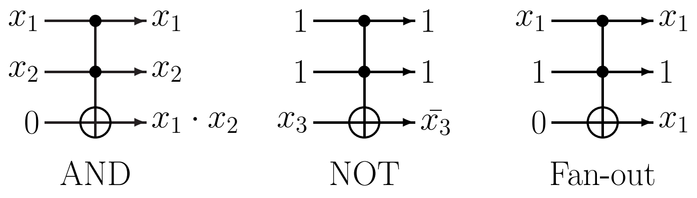


Figure 4: Implementing AND, NOT, and fan-out by Toffoli gates.

**Theorem 2.1** *(Fredkin, Toffoli [11], and Toffoli [48]) The sets {Fredkin gate} and {Toffoli gate} are both logically universal.*

It is well known that, if we can use delay elements (i.e., a memory elements) in addition to a logically universal set of gates, we can construct any sequential machine. Therefore, we can build a universal computer by using Fredkin gates or Toffoli gates, and delay elements. It should be noted that it becomes an infinite circuit to construct a storage unit like a tape of a Turing machine.

The Fredkin gate and the Toffoli gate are both 3-input 3-output gates. How are 2-input 2-output gates? Toffoli [48] showed any 2-input 2-output reversible gate is composed only of CNOTs and NOTs. Since it is easy to see that {CNOT, NOT} is not logically universal, there is no logically universal gate in 2-input 2-output reversible gates.

### 2.1.2 Garbageless reversible logic circuits

From Theorem 2.1 (and Fig. 3), we can see any logic function $f : \{0,1\}^m \to \{0,1\}^n$ can be realized by a circuit $\Phi$ composed of Fredkin gates, by allowing constant inputs $\boldsymbol{c}$ and garbage outputs $\boldsymbol{g}$ besides true inputs $\boldsymbol{x}$ and true outputs $\boldsymbol{y}$ as shown in Fig. 5. However, disposing the garbage $\boldsymbol{g}$ outside of the circuit is in fact an irreversible process. Though the circuit $\Phi$ is reversible *inside* of it, we must show how to supply constants $\boldsymbol{c}$, and how to process garbage $\boldsymbol{g}$. Otherwise, the story will not come to an end. Fredkin and Toffoli [11] showed a method to do so. (This is a logic circuit version of the garbage-less reversible Turing machine by Bennett [4] discussed in section 3.1.2. )

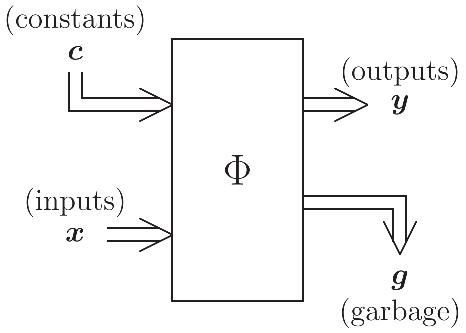


Figure 5: Embedding a logic function $f$ in a reversible logic circuit $\Phi$.

To give a garbageless logic circuit, we need the notion of an *inverse circuit* of a given reversible logic circuit $\Phi$ composed of Fredkin gates. Suppose $\Phi$ has no feedback loop. The inverse circuit $\Phi^{-1}$ is obtained by first taking the mirror image of the original circuit, and then exchange inputs and outputs of each element. Fig. 6 shows an example. It is easy to see that $\Phi^{-1}$ computes the inverse function. If we connect $\Phi$ and $\Phi^{-1}$ in series, garbage signals are erased reversibly, and we get again constants, which can be recycled. However, before doing so, the true outputs $\boldsymbol{y}$ should be copied by fan-out circuits. By this, we can obtain the outputs $\boldsymbol{y}$ without producing a large amount of garbage signals $\boldsymbol{g}$ (Fig. 7). (But, besides the true outputs $\boldsymbol{y}$, a small amount of signals $\boldsymbol{x}$ and $\bar{\boldsymbol{y}}$ are produced, which are in fact garbage in some situation.)

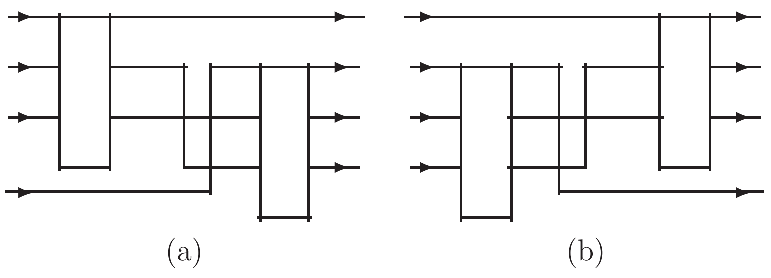


Figure 6: (a) A reversible logic circuit, and (b) its inverse circuit.

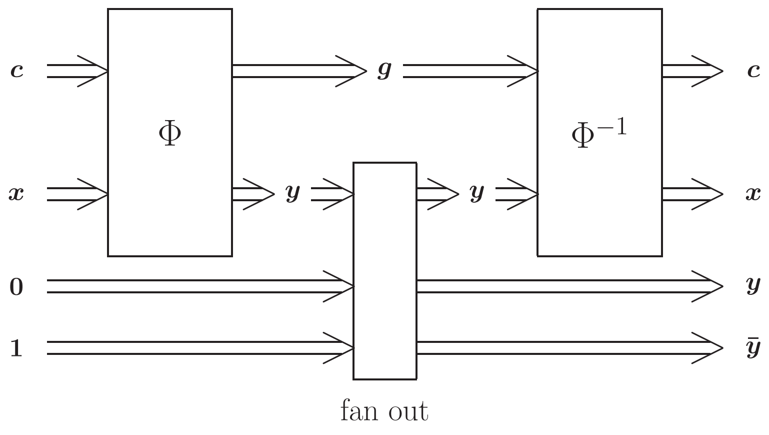


Figure 7: A garbage-less reversible logic circuits that computes the logic function $f$.

## 2.2 Reversible logic elements with memory

The traditional design theory of logic circuits mainly uses "gates" (i.e., logic elements without memory) as primitives that perform logical operations. However, as we shall see later, logic elements with memory are often useful in reversible computing. Conceptually, a gate (with two or more inputs) is an object at which "incoming" signals interact or operate on each other. Its output is a result of the interaction of the incoming signals.

Hence, some synchronization mechanism (like a clock) is necessary to make incoming signals to reach the gate at the same time. On the other hand, in the case of an element with memory, its state can be regarded as a kind of stationary signal that is always at the element's position, and can interact with an incoming signal. Therefore, if a circuit is appropriately designed, we can eliminate a clock from the circuit. Moreover, a specific kind of reversible element with memory called a "rotary element" (RE) [35] is useful for designing a circuit with a complex function. In particular, any reversible Turing machine and reversible sequential machine can be built very simply only from REs as a completely garbageless circuit.

### 2.2.1 Reversible logic elements with memory, and their classification

A reversible logic elements with memory (RLEM) is nothing but a reversible sequential machine (RSM) with small numbers of states and symbols. So, we give here definitions on a sequential machine of Mealy type (i.e., a finite automaton with an output), and its reversibility.

**Definition 2.1** *A* sequential machine *(SM) $M$ is defined by*

$$M = (Q, \Sigma, \Gamma, q_0, \delta),$$

*where $Q$ is a finite non-empty set of states, $\Sigma$ and $\Gamma$ are finite non-empty sets of input and output symbols, respectively, and $q_0 \in Q$ is an initial state. $\delta : Q \times \Sigma \to Q \times \Gamma$ is a mapping called a* move function*. An SM $M = (Q, \Sigma, \Gamma, \delta)$, where no initial state is specified, is also called an SM for convenience.*

*$M$ is called a* reversible sequential machine *(RSM) if $\delta$ is one-to-one (hence in this case $|\Sigma| \leq |\Gamma|$). In an RSM, the previous state and the input are determined uniquely from the present state and the output of $M$.*

Since there are infinitely many RSMs, we should restrict candidates of useful RLEMs somehow. Here, we consider only RLEMs with 2 states (i.e., $|Q| = 2$) and with $k$ input/output symbols (i.e., $|\Sigma| = |\Gamma| = k$) for $k = 2, 3, 4$.

Let $M = (Q, \Sigma, \Gamma, \delta)$ be a 2-state 4-symbol RLEM. Here, we assume $Q = \{q_0, q_1\}$, $\Sigma = \{a, b, c, d\}$, and $\Gamma = \{w, x, y, z\}$. The move function $\delta$ is then as follows.

$$\delta : \{q_0, q_1\} \times \{a, b, c, d\} \to \{q_0, q_1\} \times \{w, x, y, z\}$$

Since $\delta$ is one-to-one, it is specified by a permutation from the set

$$\{(q_0, w), (q_0, x), (q_0, y), (q_0, z), (q_1, w), (q_1, x), (q_1, y), (q_1, z)\}.$$

Hence, there are $8! = 40320$ RLEMs for $k = 4$. They are numbered by $0, \cdots, 40319$ in the lexicographic order of permutations. There are $6! = 720$ 2-state 3-symbol RLEMS and $4! = 24$ 2-state 2-symbol RLEMs, which are also numbered in this way [38]. To indicate $k$-symbol RLEM, the prefix "$k$-" is attached to the serial number.

**Example 2.1** *The move function of the RLEM No. 4-289 is defined by the following permutation, and is given by Table 1.*

$$((q_0, w), (q_0, x), (q_1, w), (q_1, x), (q_0, y), (q_0, z), (q_1, z), (q_1, y))$$

*We also use a pictorial representation for a 2-state RLEM as shown in Fig. 8. Note that in Fig. 8, an input signal (or a particle-like object) should be given at most one input line, because each input/output line corresponds to an input/output symbol of an RSM. Therefore, we should not confuse RLEMs with gates like the Fredkin gate where each input/output line has a binary value.*

| Present state | Input | | | |
|---|---|---|---|---|
| | $a$ | $b$ | $c$ | $d$ |
| $q_0$ | $q_0\ w$ | $q_0\ x$ | $q_1\ w$ | $q_1\ x$ |
| $q_1$ | $q_0\ y$ | $q_0\ z$ | $q_1\ z$ | $q_1\ y$ |

Table 1: The move function of the 2-state 4-symbol RLEM No. 4-289.

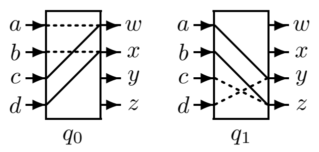


Figure 8: Pictorial representation of the RLEM No. 4-289. Solid and dotted lines in a box describe the input-output relation for each state. A solid line shows the state transits to another, and a dotted line shows the state remains unchanged.

There are many kinds of 2-state RLEMs even if we limit $k = 2, 3, 4$, but we can regard two RLEMs are "equivalent" if one can be obtained by renaming the states and the input/output symbols of the other. We can see the number of essentially different RLEMs is relatively small.

**Definition 2.2** *Let* $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1)$ *and* $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2)$ *be two LEMs.* $M_1$ *and* $M_2$ *are called* equivalent*, if there exist one-to-one onto mappings* $f : Q_1 \to Q_2$, $g : \Sigma_1 \to \Sigma_2$, *and* $h : \Gamma_1 \to \Gamma_2$ *that satisfy*

$$\forall q \in Q_1,\ \forall s \in \Sigma_1\ [\,\delta_1(q, s) = \psi(\delta_2(f(q), g(s)))\,],$$

*where* $\psi : Q_2 \times \Gamma_2 \to Q_1 \times \Gamma_1$ *is defined as follows.*

$$\forall q \in Q_2,\ \forall t \in \Gamma_2\ [\,\psi(q, t) = (f^{-1}(q), h^{-1}(t))\,]$$

The numbers of equivalence classes of 2-state 2-, 3- and 4-symbol RLEMs are 8, 24 and 82, respectively [38]. Figs. 9 and 10 show representatives of equivalence classes of 2- and 3-symbol RLEMs, where each representative is chosen as the one with the least serial number in its equivalence class.

The quotient set of 2-state $k$-symbol RLEMs can be further classified into the following three categories.

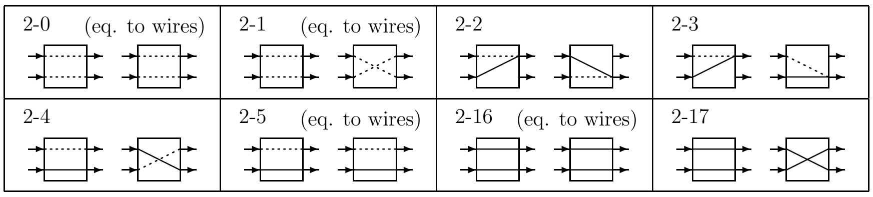


Figure 9: Representatives of 8 equivalence classes of 2-state 2-symbol RLEMs [38].

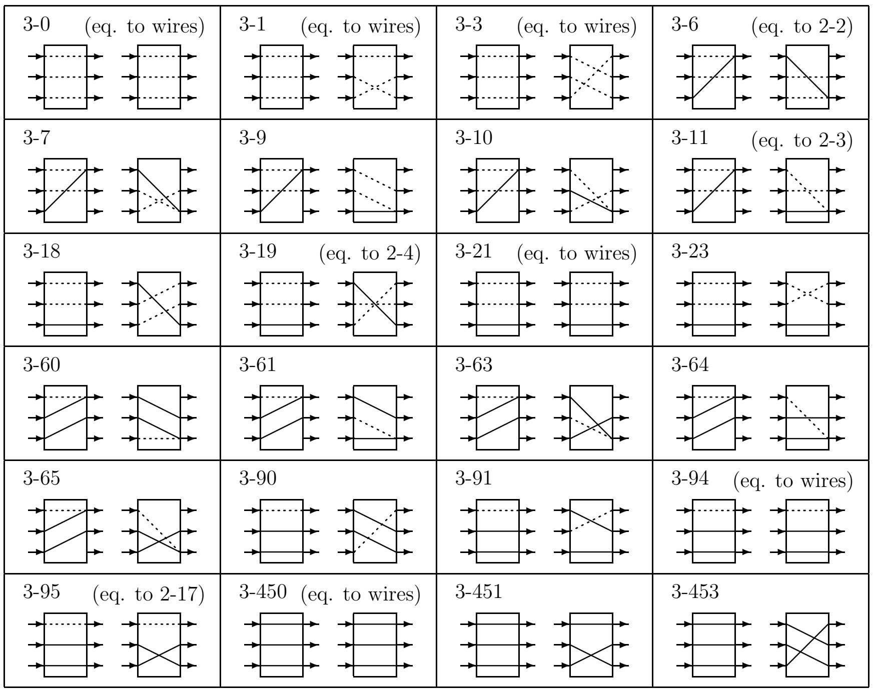


Figure 10: Representatives of 24 equivalence classes of 2-state 3-symbol RLEMs [38].

1. Elements equivalent to connecting wires:
   This is a degenerate case. For example, the element No. 3-0 in Fig. 10 acts as three wires that simply connect input and output lines. No. 3-1 is also the case, because no input makes state-change.

2. Elements equivalent to a simpler element with fewer symbols:
   This is also a degenerate case, and reducible to the elements with fewer input/output symbols. For example, the element No. 3-6 is equivalent to the element No. 2-2 plus a simple wire.

3. Proper (i.e., non-degenerate) $k$-symbol elements:
   All the RLEMs other than the cases (1) and (2) are "proper" ones.

In Figs. 9 and 10, the cases (1) and (2) are indicated at the upper-right corner of each box. Total numbers of equivalence classes of non-degenerate 2-state 2-, 3-, and 4-symbol RLEMs are 4, 14, and 55, respectively [38].

### 2.2.2 The rotary element, and logical universality of RLEMs

The *rotary element* (RE) [35] is a 2-state 4-symbol RLEM, which is equivalent to the RLEM No. 4-289 in Example 2.1, defined as follows.

$$\mathrm{RE} = (\{⊟, ◫\}, \{n, e, s, w\}, \{n', e', s', w'\}, \delta_{\mathrm{RE}})$$

The move function $\delta_{\mathrm{RE}}$ is given in Table 2.

| Present state | Input | | | |
|---|---|---|---|---|
| | $n$ | $e$ | $s$ | $w$ |
| H-state: ⊟ | ◫ $w'$ | ⊟ $w'$ | ◫ $e'$ | ⊟ $e'$ |
| V-state: ◫ | ◫ $s'$ | ⊟ $n'$ | ◫ $n'$ | ⊟ $s'$ |

Table 2: The move function $\delta_{\mathrm{RE}}$ of a rotary element RE.

The RE has the following intuitive interpretation. It has two states called H-state ( ⊟ ) and V-state ( ◫ ), and four input lines $\{n, e, s, w\}$ and four output lines $\{n', e', s', w'\}$ corresponding to the input and output alphabets (Fig. 11). We can interpret that an RE has a "rotating bar" to control the moving direction of an input signal (or a particle). When no particle exists, nothing happens on the RE. If a particle comes from a direction parallel to the rotating bar, then it goes out from the output line of the opposite side (i.e., it goes straight ahead) without affecting the direction of the bar (Fig. 12 (a)). If a particle comes from a direction orthogonal to the bar, then it makes a right turn, and rotates the bar by 90 degrees counterclockwise (Fig. 12 (b)).

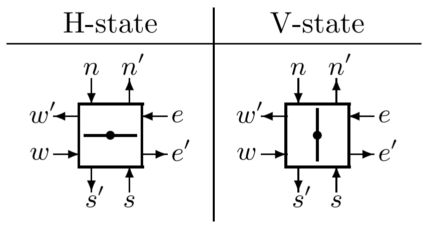


Figure 11: Two states of a rotary element (RE).

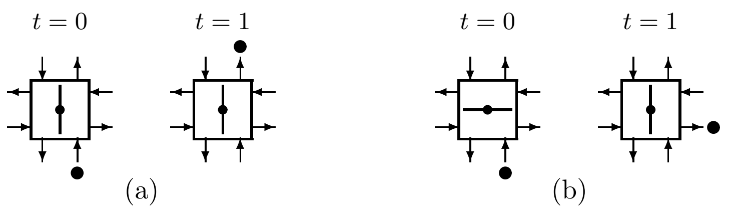


Figure 12: Operations of an RE: (a) the parallel case, and (b) the orthogonal case.

**Theorem 2.2** *(Morita, et al. [35, 38]) The set {RE} is logically universal.*

We can show this theorem by giving a circuit composed only of REs that simulates a Fredkin gate. Fig. 13 shows one such circuit. In this figure, small triangles represent delay elements, where the number written inside of them is its delay time. In this circuit, we assume all the elements works synchronously.

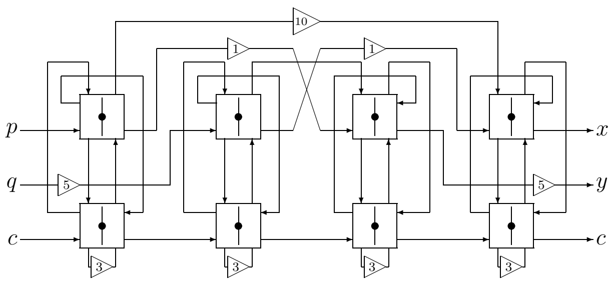


Figure 13: A realization of a Fredkin gate out of REs and delay elements [38].

It has been known that not only the RE but also each of "all" the 14 non-degenerate 2-state 3-symbol RLEMs is logically universal.

**Theorem 2.3** *(Ogiro, et al. [43]) The following 14 sets are logically universal: {No.3-7}, {No.3-9}, {No.3-10}, {No.3-18}, {No.3-23}, {No.3-60}, {No.3-61}, {No.3-63}, {No.3-64}, {No.3-65}, {No.3-90}, {No.3-91}, {No.3-451}, and {No.3-453}.*

For example, logical universality of {No.3-7} is shown by giving a circuit shown in Fig. 14 composed of RLEM No. 3-7 which realizes a Fredkin gate. In this circuit, some input/output ports of RLEMs are "open" (i.e., not connected to other ports). We assume no signal is given to the open input ports. Then, we can see that the open output ports will give no signal, by checking the operation of the circuit for each case of the inputs $c, p$, and $q$.

On the other hand, it is an open problem whether a single non-degenerate 2-state 2-symbol RLEM is logically universal or not, though {No.2-3, No.2-4} is known to be logically universal [21].

### 2.2.3 Building reversible sequential machines by reversible logic elements

As shown in Fig. 7, we can obtain an "almost" garbageless circuit made of Fredkin gates that realizes a given logic function. But, in this circuit, a small amount of garbage signals $\boldsymbol{x}$ and $\bar{\boldsymbol{y}}$ are produced at each step of operations. This causes a problem when we want to implement sequential machines. However, if we restrict constructed sequential machines to reversible ones, we can design completely garbageless circuits. Here, a "completely garbageless" means that it has no extra input/output lines other than true input/output lines, hence it is a "closed" circuit except the true input/output.

**Theorem 2.4** *(Morita [31]) For any RSM, there is a completely garbageless circuit composed only of Fredkin gates that simulates the RSM.*

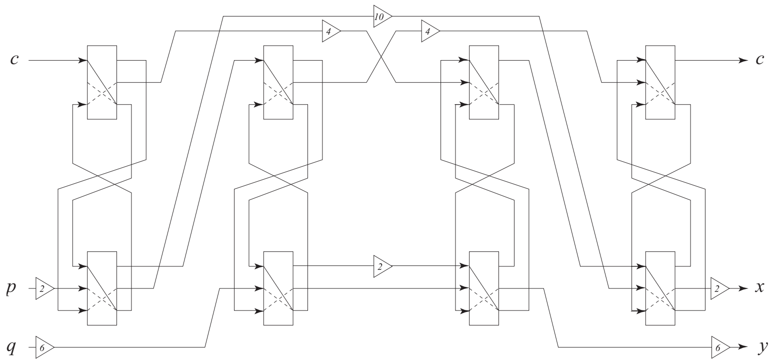


Figure 14: A realization of a Fredkin gate out of RLEMs No. 3-7 and delay elements [43].

Since the Fredkin gate can be realized by REs as in Fig. 13, we can also construct a completely garbageless circuit composed of REs. But, this is not a clever method, because several REs are needed to simulate only one Fredkin gate. There is a very simple construction method of a completely garbageless circuit composed of REs that simulates an RSM.

**Theorem 2.5** *(Morita [37]) For any RSM, there is a completely garbageless circuit composed only of REs that simulates the RSM.*

We explain the method below. First consider a circuit composed of $k + 1$ REs shown in Fig. 15, called an *RE-column*. In a resting state, each RE in the RE-column is in the vertical state except the bottom one indicated by $x$. If $x$ is in the horizontal state, the RE-column is called in the *marked* state, otherwise *unmarked* state. It can be regarded as if it is a 2-state RSM with $2k$ input symbols $\{l_1, \cdots, l_k, r_1, \cdots, r_k\}$ and $2k$ output symbols $\{l'_1, \cdots, l'_k, r'_1, \cdots, r'_k\}$, though it has many transient states. Its move function is shown in Table 3.

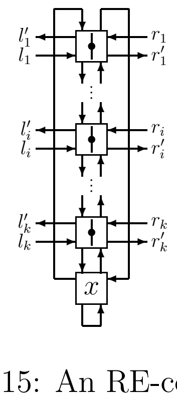


Figure 15: An RE-column.

If an RE-column is in the unmarked state and a signal is given to $l_i$, then the signal goes out from $r'_i$ without changing the state. Hence, in this case, the signal goes rightward

|  | Input |  |
|---|---|---|
| State $x$ | $l_i$ | $r_i$ |
| ⊟ (marked) | ◫ $l'_i$ | ⊟ $l'_i$ |
| ◫ (unmarked) | ◫ $r'_i$ | ⊟ $r'_i$ |

Table 3: The move function of the RE-column regarded as a 2-state RSM.

through the RE-column as if nothing happened. On the other hand, if the RE-column is in the marked state, then the signal goes out from $l'_i$ changing the state into unmarked. Therefore, if we connect many RE-columns in a row, and give a signal to $l_i$ of the leftmost RE-column, we can find the leftmost marked RE-column. This can be used as a kind of decoder to find the combination of a state and an input of an RSM.

If an RE-column is in the unmarked state and a signal is given to $r_i$, then the signal goes out from $r'_i$ changing the state to the marked one. This property can be used as a kind of encoder to set the next state of the RSM and to give an appropriate output.

|  | Input |  |
|---|---|---|
| State | $a_1$ | $a_2$ |
| $q_1$ | $q_2 b_1$ | $q_3 b_2$ |
| $q_2$ | $q_2 b_2$ | $q_1 b_1$ |
| $q_3$ | $q_1 b_2$ | $q_3 b_1$ |

Table 4: A move function $\delta_1$ of an RSM $M_1$.

We consider an example of an RSM $M_1 = (\{q_1, q_2, q_3\}, \{a_1, a_2\}, \{b_1, b_2\}, \delta_1)$, and show how to make a completely garbageless circuit that simulates $M_1$. Its move function $\delta_1$ is given in Table 4. Prepare three RE-columns with $k = 2$, each of which corresponds to each state of $M_1$, and connect them as shown in Fig. 16. The leftmost input lines $l_1$ and $l_2$ correspond to the input symbols $a_1$ and $a_2$. The rightmost output lines $r'_1$ and $r'_2$ correspond to the output symbols $b_1$ and $b_2$. The output line $l'_i$ of each column is connected to an input line $r_j$ of an appropriately chosen RE-column according to $\delta_1$. If $M_1$ is in the state $q_1$, then only the first RE-column is set to the marked state. If an input signal comes from $a_1$, then the signal first goes out from $l'_1$ of the first RE-column, setting this column to the unmarked state. Then, this signal enters the second column from $r_1$. This makes this column marked, and finally goes out from the output line $b_1$, which realizes $\delta_1(q_1, a_1) = (q_2, b_1)$. Likewise, if $M_1$ is in $q_1$ and a signal is given to $a_2$, then it goes out from $l'_2$ of the first column, setting this column unmarked. The signal enters the third column from $r_2$, making this column marked. Finally it goes out from $b_2$, which realizes $\delta_1(q_1, a_2) = (q_3, b_2)$. Note that, even if each RE operates asynchronously, the circuit works correctly, because only one signal exists when it is operating.

It is easily seen that the above method can be generalized to construct any RSM by REs. In fact, any RSM can be realized by decoding, permuting, and encoding operations, and the above circuit composed of REs simply does them.

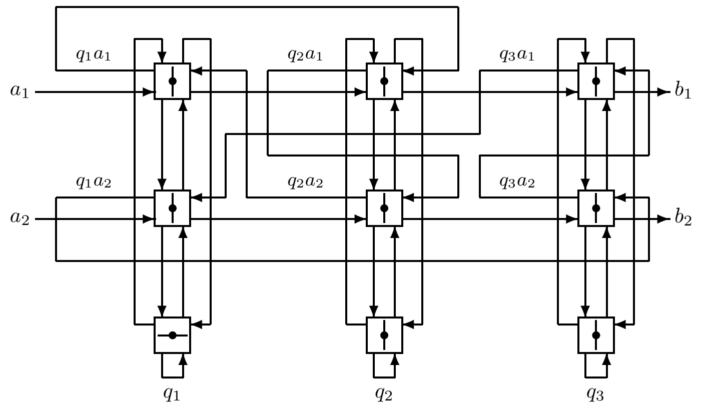


Figure 16: A garbage-less circuit made of REs that simulates an RSM $M_1$.

## 2.3 Realization by the Billiard Ball Model (BBM)

The Billiard Ball Model (BBM) is a reversible physical model of computing proposed by Fredkin and Toffoli [11]. It is an idealized mechanical model consisting of balls and reflectors (Fig. 17). Balls can collide with other balls or reflectors. It is assumed that collisions are elastic, and there is no friction. Although it is impossible to realize BBM as a real hardware, it is very insightful that a universal computer can be constructed in such a manner. Margenstern [22] showed this realization requires an infinite initial configuration.

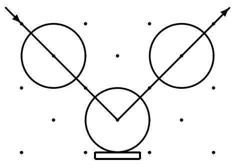


Figure 17: The Billiard Ball Model (BBM).

### 2.3.1 BBM realization of the Fredkin gate

The Fredkin gate is realizable in BBM in the following way [11]. Firstly, we need other reversible logic gates called the *switch gate* and its inverse. It is a 2-input 3-output gate shown in Fig. 18, which realizes the one-to-one logic function $(c, x) \mapsto (c, cx, \bar{c}x)$. The inverse switch gate is a 3-input 2-output gate whose logic function is defined only on $\{(0,0,0),(0,0,1),(1,0,0),(1,1,0)\}$ (otherwise, it cannot be a reversible gate), and it is the inverse function of the switch gate. The Fredkin gate can be built from switch gates and inverse switch gates as in Fig. 19 (in [11] it is noted that this circuit is due to R. Feynman and A. Ressler). Fig. 20 shows how the switch gate is realized in BBM. Therefore, the Fredkin gate is also implemented in BBM.

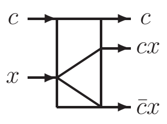


Figure 18: The switch gate.

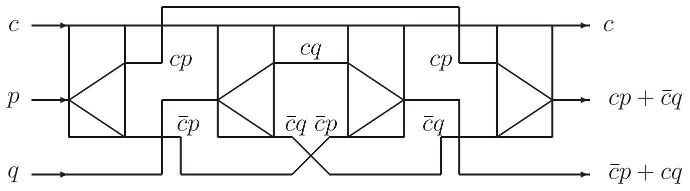


Figure 19: The Fredkin gate made of two switch gates and two inverse switch gates.

### 2.3.2 BBM realization of the rotary element

Here, we show a new realization method of an RE in BBM, which is much simpler than the case of the Fredkin gate. Fig. 21 gives a configuration of BBM that can simulate an RE. It consists of one stationary ball, and many reflectors that are shown by small rectangles. If a stationary ball is put at the position V (or H, respectively), then we regard it is in the V-state (H-state). In Fig. 21, trajectories of balls for all the cases of operations are written.

Fig. 22 shows the case that the RE is in the V-state and the input signal is given to $s$ (it corresponds to the case (a) in Fig. 12). In this case, incoming (i.e., dark) ball simply goes straight ahead interacting neither with the stationary (i.e., white) ball nor with reflectors, and finally comes out from the output line $n'$. This is the trivial case.

Fig. 23 shows the case that the RE is in the H-state and the input signal is given to $s$ (it corresponds to the case (b) in Fig. 12). The dark ball collides with the white ball, and they travel along the trajectories $s_0$ and $s_1$. Then the balls again collide. By this, the white one stops at the position V, while the dark one travels rightward and finally goes out from $e'$. Note that the second collision is a reverse process of the first collision.

An advantage of this method is that, in a resting state, the ball in this configuration is stationary. Therefore, the incoming ball can reach it at any moment and with any velocity, hence there is no need to synchronize the two balls. Therefore, it can be used in an “asynchronous” mode as discussed in section 2.2.3. In the case of the BBM realization of the Fredkin gate shown in the previous section, a lot of balls must be synchronized precisely, and have exactly the same velocity. It is indeed extremely difficult.

One may think it is still impossible to implement an RE by a real mechanical hardware of BBM. However, considering the role of the stationary ball in this method, it need not be a movable particle, but can be a state of some stationary object, such as a quantum state of an atom or like that. The incoming ball can be a moving particle something like a photon, and there may be no need to adjust the arrival timing. So, this method suggests that there will be a possibility to implement an RE in a real physical system with reversibility.

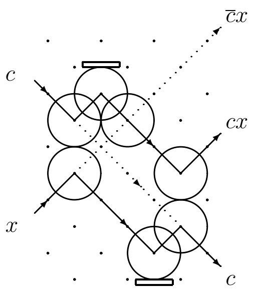


Figure 20: A BBM realization of the switch gate [11].

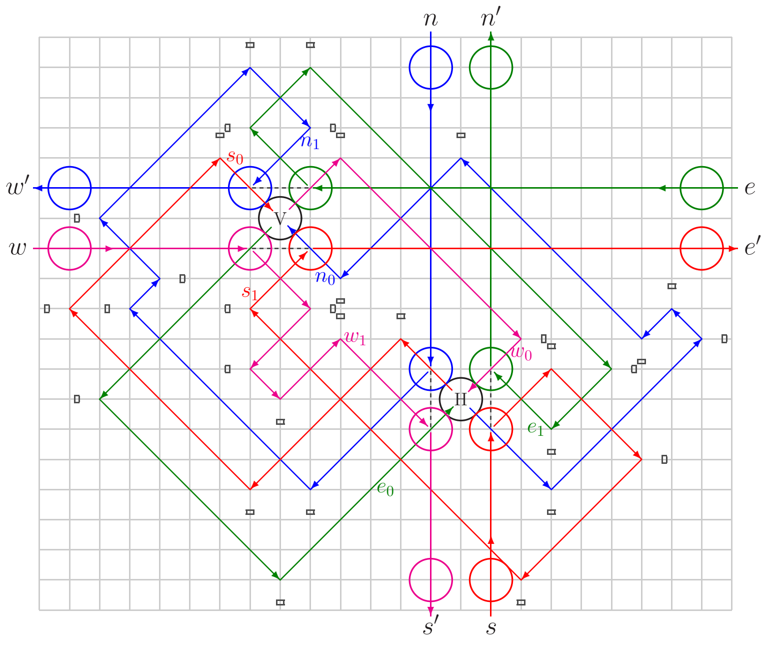


Figure 21: A rotary element (RE) realized in BBM.

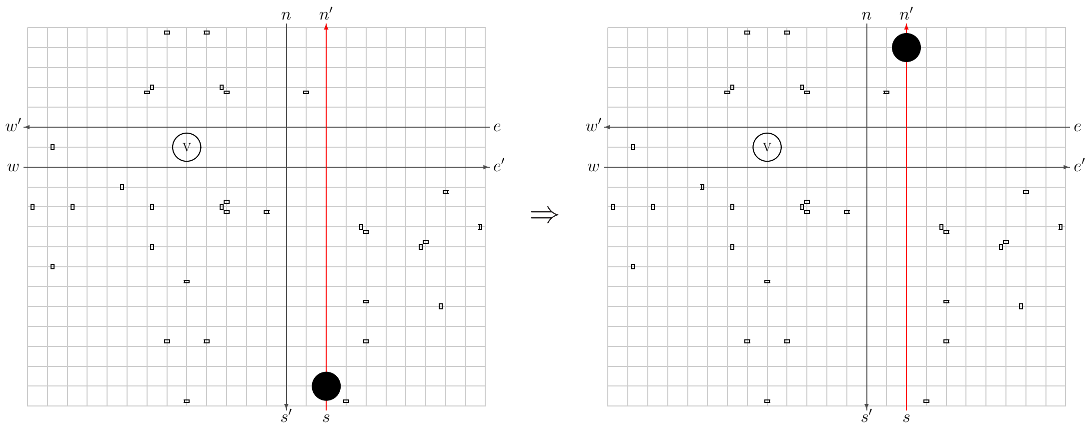


Figure 22: Simulating the parallel case of the operation of an RE. It corresponds to the case (a) in Fig. 12.

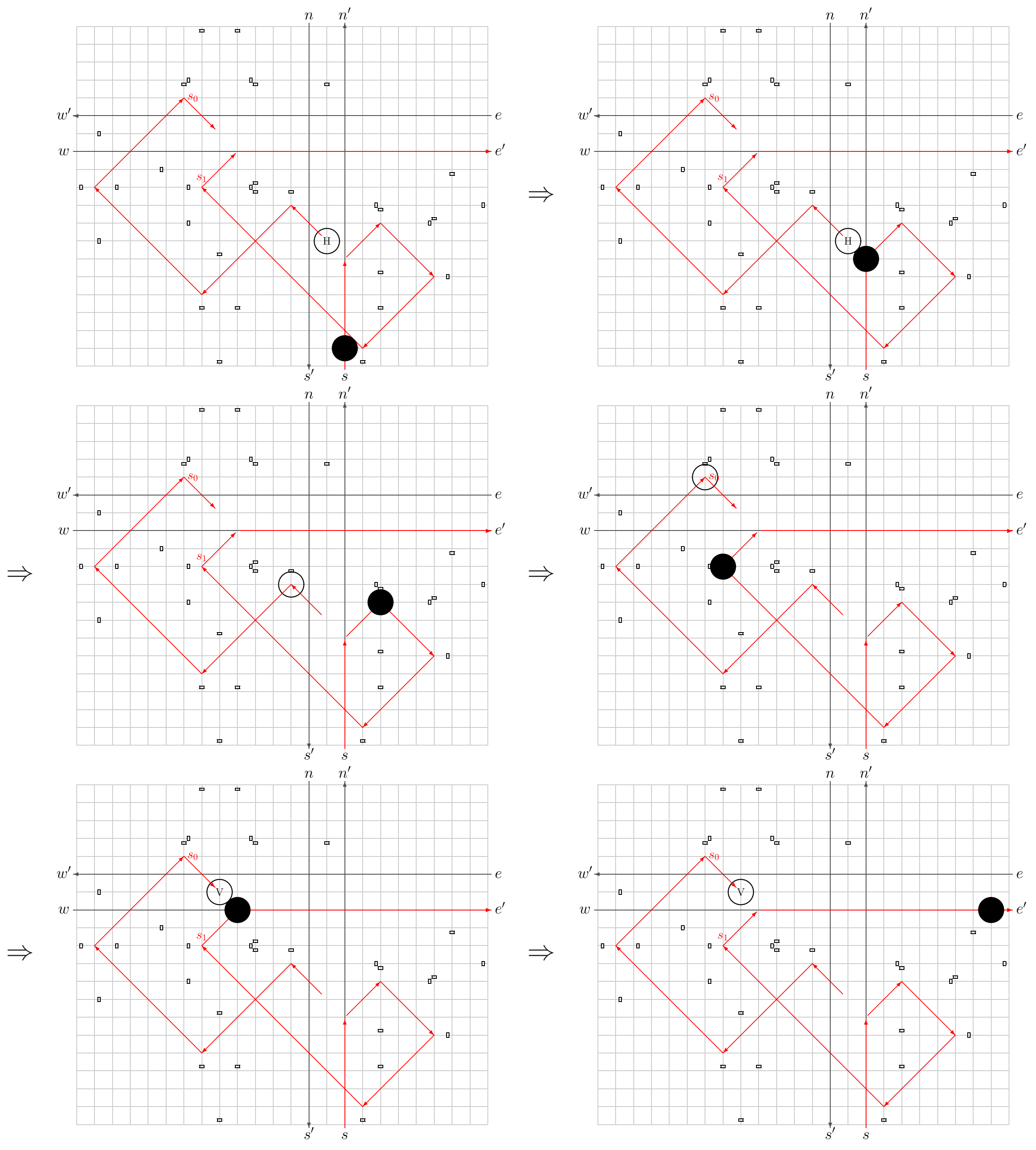


Figure 23: Simulating the orthogonal case of the operation of an RE. It corresponds to the case (b) in Fig. 12.

# 3 Reversible Turing machines

In this section, we discuss reversibility in computing mainly from the system level rather than the element level. Since a Turing machine is the standard model in the traditional theory of computing, it is convenient to study first a reversible Turing machine (RTM) among various reversible systems. Lecerf [20] first investigated RTMs, and showed unsolvability of the halting problem and some related problems on them. Bennett [4, 5, 6] studied them from the standpoint of thermodynamics of computing. He showed that every irreversible Turing machine (TM) can be simulated by an RTM with three tapes without leaving garbage information on the tapes when it halts. Hence, garbageless RTMs are computation-universal. We shall also see that RTMs can be built from reversible logic elements. In particular, Morita [35] showed that any RTM can be implemented very simply as a garbageless circuit made of REs. Finally, we show that there are rather small universal RTMs [39].

## 3.1 Reversible Turing machines and their universality

### 3.1.1 Definitions on reversible Turing machines

When Bennett [4] introduced an RTM, a quadruple formulation was used, because an "inverse" TM for a given RTM is easily defined. He used it to obtain an RTM that performs garbageless computation.

**Definition 3.1** *A* 1-tape Turing machine in the quadruple form *is defined by*

$$T_4 = (Q, S, q_0, q_f, s_0, \delta),$$

*where $Q$ is a non-empty finite set of states, $S$ is a non-empty finite set of symbols, $q_0$ is an initial state ($q_0 \in Q$), $q_f$ is a final (halting) state ($q_f \in Q$), $s_0$ is a special blank symbol ($s_0 \in S$). $\delta$ is a move relation, which is a subset of $(Q \times S \times S \times Q) \cup (Q \times \{/\} \times \{-, 0, +\} \times Q)$. Each element of $\delta$ is a quadruple of the form either $[p, s, s', q] \in (Q \times S \times S \times Q)$ or $[p, /, d, q] \in (Q \times \{/\} \times \{-, 0, +\} \times Q)$. The symbols "$-$", "0", and "$+$" stand for "left-shift", "zero-shift", and "right-shift", respectively. $[p, s, s', q]$ means if $T_4$ reads the symbol $s$ in the state $p$, then write $s'$ and go to the state $q$. $[p, /, d, q]$ means if $T_4$ is in $p$, then shift the head to the direction $d$ and go to the state $q$.*

*$T_4$ is called a* deterministic Turing machine *iff the following condition holds for any pair of distinct quadruples $[p_1, b_1, c_1, q_1]$ and $[p_2, b_2, c_2, q_2]$ in $\delta$. (Since we deal with only deterministic TMs here, we omit the word "deterministic" hereafter.)*

$$\text{If } p_1 = p_2, \text{ then } b_1 \neq / \ \wedge \ b_2 \neq / \ \wedge \ b_1 \neq b_2.$$

*$T_4$ is called a* reversible Turing machine *(RTM) iff the following condition holds for any pair of distinct quadruples $[p_1, b_1, c_1, q_1]$ and $[p_2, b_2, c_2, q_2]$ in $\delta$.*

$$\text{If } q_1 = q_2, \text{ then } b_1 \neq / \ \wedge \ b_2 \neq / \ \wedge \ c_1 \neq c_2.$$

It is easy to see that, if a TM satisfies the above definition of reversibility, then every computational configuration of the TM has at most one predecessor. On the other hand, if there is a pair of distinct quadruples that does not satisfy the above condition, then we can easily find a pair of distinct computational configurations that transit to the same configuration at the next time step. Hence, the above definition exactly characterizes the notion of an "injective" TM where every computational configuration has at most one predecessor.

Definition 3.1 can be extended to multi-tape RTMs. For example, a 2-tape TM is defined by

$$T = (Q, (S_1, S_2), q_0, q_f, (s_{1,0}, s_{2,0}), \delta).$$

A quadruple in $\delta$ is either of the form $[p, [s_1, s_2], [s_1', s_2'], q] \in (Q \times (S_1 \times S_2) \times (S_1 \times S_2) \times Q)$ or of the form $[p, /, [d_1, d_2], q] \in (Q \times \{/\} \times \{-, 0, +\}^2 \times Q)$. Reversibility is defined as follows, which is similar as above. $T$ is called *reversible* iff the following condition holds for any pair of distinct quadruples $[p_1, X_1, [b_1, b_2], q_1]$ and $[p_2, X_2, [c_1, c_2], q_2]$ in $\delta$.

$$\text{If } q_1 = q_2, \text{ then } X_1 \neq / \ \wedge \ X_2 \neq / \ \wedge \ [b_1, b_2] \neq [c_1, c_2].$$

Next, we give a quintuple formulation of RTMs compatible with the quadruple formulation. Because, classical TMs are usually defined in the quintuple formulation, and hence convenient to compare with them (e.g., comparing the sizes of universal TMs and universal RTMs).

**Definition 3.2** *A* 1-tape Turing machine in the quintuple form *is defined by*

$$T_5 = (Q, S, q_0, q_f, s_0, \delta),$$

*where $Q, S, q_0, q_f, s_0$ are the same as in Definition 3.1. $\delta$ is a move relation, which is a subset of $(Q \times S \times S \times \{-, 0, +\} \times Q)$. Each element of $\delta$ is a quintuple of the form $[p, s, s', d, q]$. It means if $T_5$ reads the symbol $s$ in the state $p$, then write $s'$, shift the head to the direction $d$, and go to the state $q$.*

*$T_5$ is called* deterministic *iff the following condition holds for any pair of distinct quintuples $[p_1, s_1, s_1', d_1, q_1]$ and $[p_2, s_2, s_2', d_2, q_2]$ in $\delta$.*

$$\textit{If } p_1 = p_2, \textit{ then } s_1 \neq s_2.$$

*$T_5$ is called* reversible *iff the following condition holds for any pair of distinct quintuples $[p_1, s_1, s_1', d_1, q_1]$ and $[p_2, s_2, s_2', d_2, q_2]$ in $\delta$.*

$$\textit{If } q_1 = q_2, \textit{ then } s_1' \neq s_2' \ \wedge \ d_1 = d_2.$$

The above definition of reversibility says that, for any pair of quintuples, if the next states are the same, then the shift directions should be the same and the written symbols should be different. Therefore, also in the quintuple case, if the above condition holds, then each computational configuration of the TM has at most one predecessor, and vice versa. Thus, it exactly characterizes an injective TM. Furthermore, it is also easy to show that any RTM in the quintuple form can be simulated by an RTM in the quadruple form, and vice versa [39]. Hence, in this sense, these two definitions of reversibility are equivalent.

### 3.1.2 Universality of garbageless reversible Turing machines

When given an irreversible TM, it is relatively easy to construct an RTM that simulates the former. The RTM simply writes down *all* the history of the computation process of the irreversible TM on an additional tape. By using this information the RTM can go back its computing process at any time. But, it leaves a large amount of garbage information besides the true output on the tape when it halts. Discarding the garbage is, in fact, an irreversible operation, hence we should give a method of dealing with garbage information.

**Theorem 3.1** *(Bennett [4]) For any irreversible 1-tape TM $T$, we can construct a garbage-less 3-tape RTM $T'$ that simulates the former and gives only an input string and an output string on the tapes when it halts.*

An outline of the RTM $T'$ is as follows. The first tape is a working tape on which the tape of $T$ is simulated. The second tape is a history tape on which the number of the executed quadruple (or quintuple) of $T$ is written at each simulation step. The third tape is an output tape on which the result of the computation by $T$ is copied. $T'$ first simulates $T$ recording the computation history on the history tape. If $T$ halts, then $T'$ copies the result on the working tape to the output tape. Then $T'$ performs the "backward computation" reversibly erasing the history tape.

**Example 3.1** *Let us consider the irreversible TM $T_{\text{erase}}$.*

$$\begin{aligned} T_{\text{erase}} &= (\{q_0, q_1, q_2, q_3, q_4, q_f\}, \{0,1,2\}, q_0, q_f, 0, \delta_{\text{erase}}) \\ \delta_{\text{erase}} &= \{\ 1 : [q_0, /, +, q_1],\ \ 2 : [q_1, 0, 0, q_4],\ \ 3 : [q_1, 1, 1, q_2],\ \ 4 : [q_1, 2, 1, q_2], \\ &\qquad 5 : [q_2, /, +, q_1],\ \ 6 : [q_3, 0, 0, q_f],\ \ 7 : [q_3, 1, 1, q_4],\ \ 8 : [q_4, /, -, q_3]\ \} \end{aligned}$$

*If an input string consisting of 1's and 2's is given to $T_{\text{erase}}$, it changes all 2's into 1's, and halts as in Fig. 24). The RTM $T'_{\text{erase}}$ simulates $T_{\text{erase}}$ with the input 122 as shown in Fig. 25.*

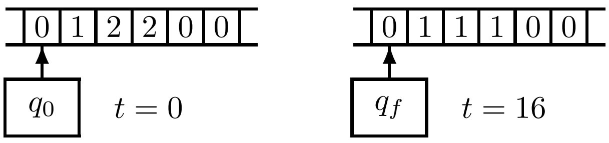


Figure 24: The initial and the final configuration of $T_{\text{erase}}$ for the input 122.

The RTM in Theorem 3.1 uses large amount of space, which is proportional to the time that the simulated TM uses. Bennett [7] showed a method for reducing space in the following space-time trade-off theorem.

**Theorem 3.2** *(Bennett [7]) For any $\varepsilon > 0$, and for any irreversible TM using time $T$ and space $S$, we can construct an RTM that simulates the former in time $O(T^{1+\varepsilon})$ and space $O(S \log T)$.*

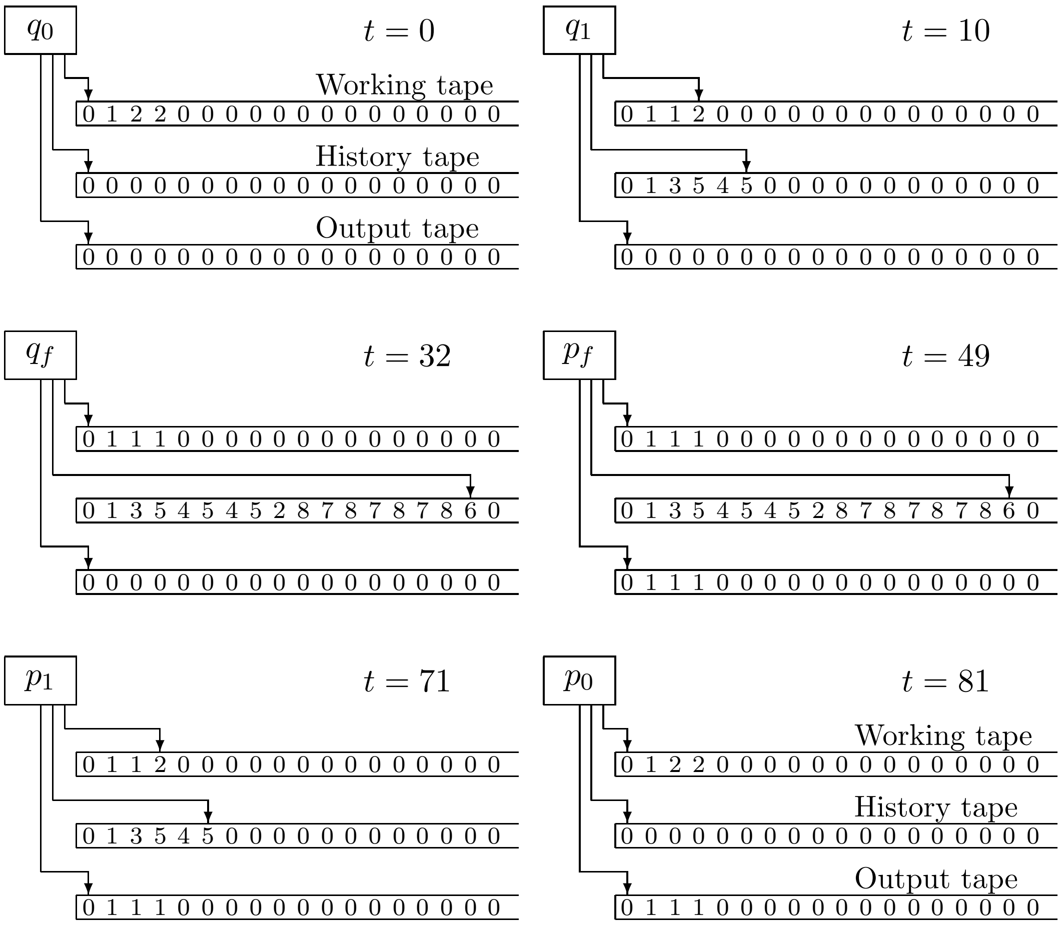


Figure 25: Simulating $T_{\text{erase}}$ with the input 122 by the garbageless RTM $T'_{\text{erase}}$: the forward computation stage (from $t = 0$ to $t = 32$), the copying stage (from $t = 33$ to $t = 48$), and the backward computation stage (from $t = 49$ to $t = 81$).

## 3.2 Building reversible Turing machines by rotary elements

We saw any RSM can be constructed by using REs as a completely garbageless circuit (Theorem 2.5). Since a finite-state control and a tape cell of an RTM are formalized as RSMs, it is possible to construct any RTM by REs. A detailed construction method is shown in [35]. Here, we only give an example.

**Example 3.2** *Consider an RTM* $T_{\text{parity}} = (Q, \{0,1\}, q_0, q_a, 0, \delta)$, *where* $Q = \{q_0, q_1, q_2, q_3, q_4, q_a, q_r\}$, *and* $\delta$ *is as below.*

$$\begin{aligned}\delta = \{ \quad & [\ q_0, 0, 1, q_1\ ],\quad [\ q_1, /, +, q_2\ ],\quad [\ q_2, 0, 1, q_a\ ],\quad [\ q_2, 1, 0, q_3\ ],\\ & \qquad\qquad\qquad\quad\ [\ q_3, /, +, q_4\ ],\quad [\ q_4, 0, 1, q_r\ ],\quad [\ q_4, 1, 0, q_1\ ]\ \}.\end{aligned}$$

$T_{\text{parity}}$ *checks if a given unary number* $n$ *is even or odd. If it is even,* $T_{\text{parity}}$ *halts in the accepting state* $q_a$, *otherwise halts in the state* $q_r$. *All the symbols read by* $T_{\text{parity}}$ *are complemented as in Fig. 26. A garbageless reversible logic circuit composed only of REs that simulates* $T_{\text{parity}}$ *is shown in Fig. 27.*

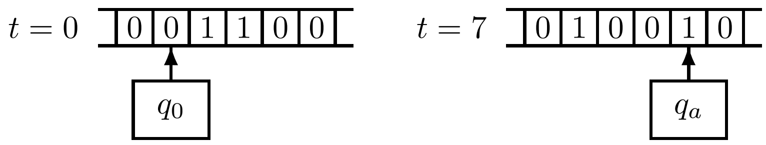


Figure 26: The initial and the final configuration of $T_{\text{parity}}$ for a given unary input 11.

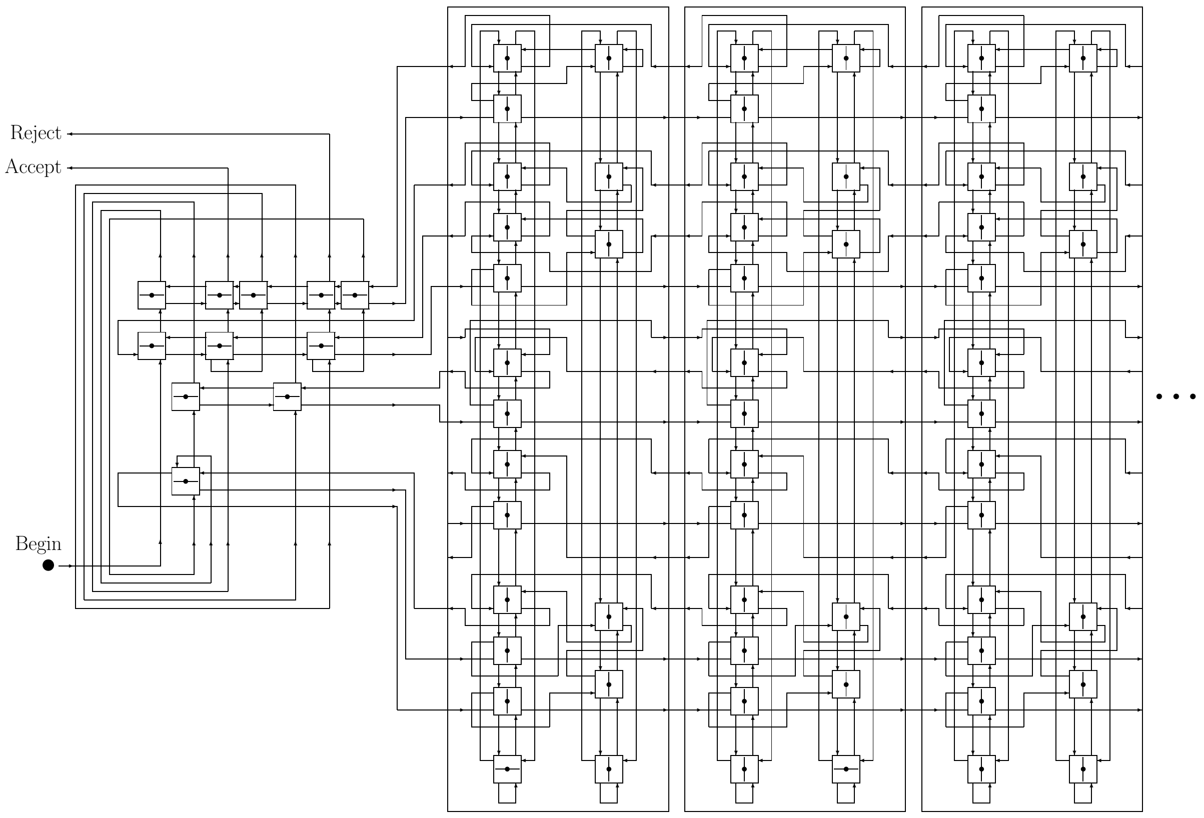


Figure 27: A garbageless circuit composed only of REs that simulates the RTM $T_{parity}$

*There is a finite-state control in the left part of Fig. 27 . To the right of it, infinitely many copies of a tape cell module are connected. This circuit is closed except the Begin, Accept, and Reject ports. Setting the initial states of tape cell modules, and then giving a signal from Begin port, it starts to compute. If the RTM halts in the accepting state $q_a$ (or rejecting state $q_r$, respectively), a signal goes out from the Accept (Reject) port, leaving an answer in the tape cell modules.*

*This circuit is further implemented in BBM by the method shown in section 2.3.2. In this case, many stationary balls are put at appropriate positions in a circuit composed of a lot of reflectors. Then, a moving ball is given to the Begin port at any time and in any speed. When the computation terminates, the moving ball will go out from the Accept or the Reject port.*

## 3.3 A small universal reversible Turing machine

By Theorem 3.1, we know the existence of a *universal RTM* (URTM) that can compute any recursive function. But, if we convert a classical (irreversible) UTM to a URTM, its size becomes large. To obtain a small URTM, we need some additional framework. In the classical case, most small UTMs simulates a 2-tag system [26], a kind of string rewriting system with computation-universality. In the case of RTM, a 17-state 5-symbol URTM is obtained by Morita and Yamaguchi [39] by simulating a cyclic tag system (CTAG) defined by Cook [9], which is also a very simple string rewriting system with computation-universality. Note that Woods and Neary [51] also use CTAGs to design small semi-weakly UTMs.

### 3.3.1 Cyclic tag systems

In the original definition of a cyclic tag system by Cook [9], the notion of halting was not defined explicitly. To deal with halting of a URTM, we use here a modified definition of it that can simulate a 2-tag system with the halting property [39].

**Definition 3.3** *A* cyclic tag system *(CTAG) is defined by*

$$C = (k, \{Y, N\}, (\text{halt}, p_1, \cdots, p_{k-1})),$$

*where $k$ ($k = 1, 2, \cdots$ ) is the length of a cycle (i.e., period), $\{Y, N\}$ is the (fixed) alphabet, and $(p_1, \cdots, p_{k-1}) \in (\{Y, N\}^*)^{k-1}$ is a $(k-1)$-tuple of production rules. A pair $(v, m)$ is called an* instantaneous description *(ID) of $C$, where $v \in \{Y, N\}^*$ and $m \in \{0, \cdots, k-1\}$. $m$ is called a* phase *of the ID. A* halting ID *is an ID of the form $(Yv, 0)$ ($v \in \{Y, N\}^*$). The transition relation $\Rightarrow$ is defined as follows. For any $(v, m), (v', m') \in \{Y, N\}^* \times \{0, \cdots, k-1\}$,*

$$\begin{array}{lll} (Yv, m) \Rightarrow (v', m') & \textit{iff} & [m \neq 0] \wedge [m' = m + 1 \bmod k] \wedge [v' = vp_m], \\ (Nv, m) \Rightarrow (v', m') & \textit{iff} & [m' = m + 1 \bmod k] \wedge [v' = v]. \end{array}$$

*A sequence of IDs $((v_0, m_0), \cdots, (v_n, m_n))$ is called a* complete computation starting from *$v \in \{Y, N\}^*$ iff $(v_0, m_0) = (v, 0)$, $(v_i, m_i) \Rightarrow (v_{i+1}, m_{i+1})$ ($i = 0, 1, \cdots, n-1$), and $(v_n, m_n)$ is a halting ID.*

In a CTAG, rewriting of a string is performed as follows. Assume the present phase is $m (\neq 0)$. If the first symbol of the string is $Y$, then remove it, and attach $p_m$ at the end of the string. If it is $N$, then simply remove it. If $m = 0$ and the first symbol is $Y$, then halt.

**Example 3.3** *Consider the CTAG $C_1 = (3, \{Y, N\}, (\text{halt}, YN, YYN))$. The complete computation starting from $NYY$ is $(NYY,\ 0) \Rightarrow (YY,\ 1) \Rightarrow (YYN,\ 2) \Rightarrow (YNYYN,\ 0)$.*

### 3.3.2 Simulating cyclic tag systems by a reversible Turing machine

The history of the study of universal Turing machines (UTMs) is long. So far the following (irreversible) UTMs have been found, where UTM($m, n$) denotes an $m$-state $n$-symbol UTM: a UTM(7,4) by Minsky [26], a UTM(24,2), a UTM(10,3), a UTM(7,4), a UTM(5,5), a UTM(4,6), a UTM(3,10) and a UTM(2,18) by Rogozhin [46], a UTM(19,2) by Baiocchi [3], a UTM(3,9) by Kudlek and Rogozhin [18], a UTM(5,5), a UTM(6,4), a UTM(9,3), and a UTM(18,2) by Neary and Woods [42], etc.

As for URTMs, Morita and Yamaguchi [39] showed that there is a 17-state 5-symbol URTM (URTM(17,5)) with 67 quintuples. Here, we newly show a URTM(15,6) with 62 quintuples that is constructed by a similar technique as in the URTM(17,5).

**Theorem 3.3** *There is a 15-state 6-symbol URTM (URTM(15,6)) with 62 quintuples that can simulate any CTAG.*

A URTM(15,6) $T_{15,6}$ that simulates any CTAG is as follows.

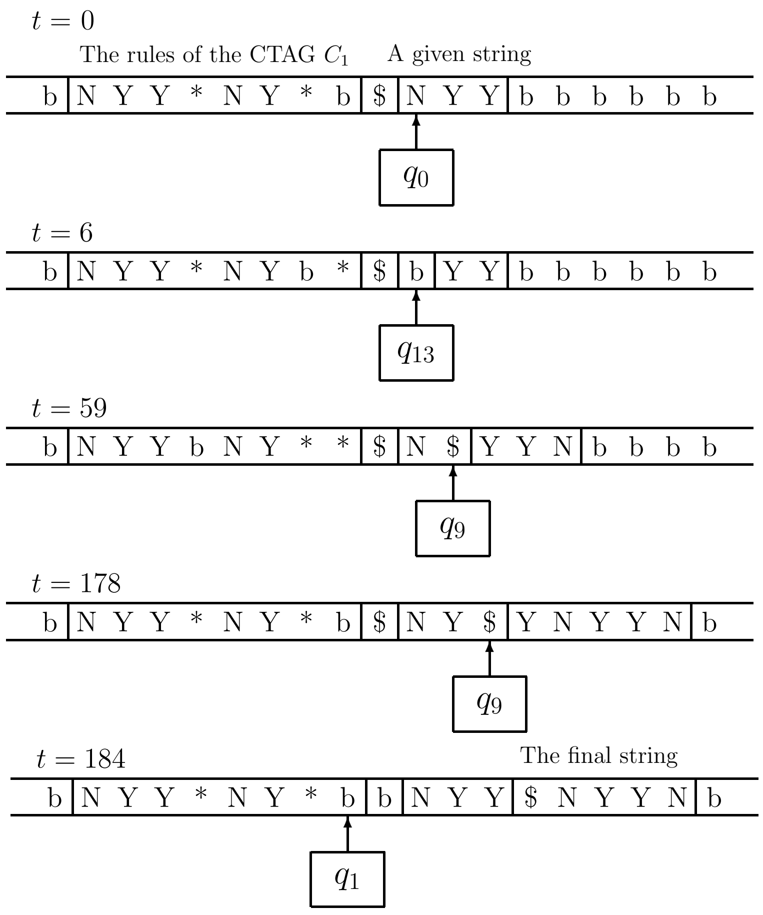
$$T_{15,6} = (\{q_0, \cdots, q_{14}\}, \{b, Y, N, *, \$, 1\}, q_0, b, \delta),$$

where the set $\delta$ of quintuples is shown in Table 5. (Note that, generally, the final state is usually omitted from the state set in the construction of a UTM.) If a CTAG halts with a halting ID, then $T_{15,6}$ halts in the state $q_1$. If the string becomes an empty string, then it halts in the state $q_2$. In Table 5, it is indicated by "null".

| | $b$ | $Y$ | $N$ | $*$ | $\$$ | 1 |
|---|---|---|---|---|---|---|
| $q_0$ | $\$-q_2$ | $\$-q_1$ | $b-q_{11}$ | | | |
| $q_1$ | halt | $Y-q_1$ | $N-q_1$ | $*+q_0$ | $b-q_1$ | |
| $q_2$ | $*-q_3$ | $Y-q_2$ | $N-q_2$ | $*-q_2$ | null | |
| $q_3$ | $b+q_{10}$ | $1+q_4$ | $b+q_5$ | $b+q_8$ | | |
| $q_4$ | $Y+q_6$ | $Y+q_4$ | $N+q_4$ | $*+q_4$ | $\$+q_4$ | |
| $q_5$ | $N+q_6$ | $Y+q_5$ | $N+q_5$ | $*+q_5$ | $\$+q_5$ | |
| $q_6$ | $b-q_7$ | | | | | |
| $q_7$ | $N-q_3$ | $Y-q_7$ | $N-q_7$ | $*-q_7$ | $\$-q_7$ | $Y-q_3$ |
| $q_8$ | | $Y+q_8$ | $N+q_8$ | $*+q_8$ | $\$+q_9$ | |
| $q_9$ | | $Y+q_9$ | $N+q_9$ | $*+q_9$ | $Y+q_0$ | |
| $q_{10}$ | | $Y+q_{10}$ | $N+q_{10}$ | $*+q_{10}$ | $\$-q_3$ | |
| $q_{11}$ | $*-q_{12}$ | $Y-q_{11}$ | $N-q_{11}$ | $*-q_{11}$ | $\$-q_{11}$ | |
| $q_{12}$ | $b+q_{14}$ | $Y-q_{12}$ | $N-q_{12}$ | $b+q_{13}$ | | |
| $q_{13}$ | $N+q_0$ | $Y+q_{13}$ | $N+q_{13}$ | $*+q_{13}$ | $\$+q_{13}$ | |
| $q_{14}$ | | $Y+q_{14}$ | $N+q_{14}$ | $*+q_{14}$ | $\$-q_{12}$ | |

Table 5: The set of quintuples of the URTM $T_{15,6}$.

Fig. 28 shows how the CTAG $C_1$ with the initial string $NYY$ in Example 3.3 is simulated by the URTM $T_{15,6}$. On the tape of the URTM, the production rules (halt, $NY$, $YYN$) of $C_1$ are expressed by the reversal sequence over $\{Y, N, *\}$, i.e., $NYY*YN**$, where $*$



$t = 0$
The rules of the CTAG $C_1$
A given string
b N Y Y * N Y * b $ N Y Y b b b b b b
$q_0$
$t = 6$
b N Y Y * N Y b * $ b Y Y b b b b b b
$q_{13}$
$t = 59$
b N Y Y b N Y * * $ N $ Y Y N b b b b
$q_9$
$t = 178$
b N Y Y * N Y * b $ N Y $ Y N Y Y N b
$q_9$
$t = 184$
The final string
b N Y Y * N Y * b b N Y Y $ N Y Y N b
$q_1$

Figure 28: Simulating the CTAG $C_1$ in Example 3.3 by the URTM $T_{15,6}$.

is used as a delimiter between rules, and "halt" is represented by the empty string. Note that in the initial tape of $T_{15,6}$ ($t = 0$), the rightmost $*$ is replaced by $b$. This indicates that the phase is 0. In general, if the phase is $i - 1$, then the $i$-th $*$ from the right is replaced by $b$. This symbol $b$ is called a "phase marker." On the other hand, the given initial string for $C_1$ is placed to the right of the rules, where \$ is used as a delimiter.

The IDs in the complete computation $(NYY,\ 0) \Rightarrow (YY,\ 1) \Rightarrow (YYN,\ 2) \Rightarrow (YNYYN,\ 0)$ of $C_1$ appear in the computational configurations of $T_{15,6}$ at $t = 0, 6, 59$ and 178, respectively. The symbol \$ in the final string ($t = 184$) should be regarded as $Y$.

It is easy to see that $T_{15,6}$ is reversible by checking the set of quintuples shown in Table 5 according to the definition of an RTM. Intuitively, its reversibility is guaranteed from the fact that no information is erased in the whole simulation process. Furthermore, every branch of the program caused by reading the symbol $Y$ or $N$ is "merged reversibly" by writing the original symbol. For example, the states $q_9$ and $q_{13}$ transit to the same state $q_0$ by writing $Y$ and $N$, respectively, using the quintuples $[q_9, \$, Y, +, q_0]$ and $[q_{13}, b, N, +, q_0]$. Although there are many open boxes in Table 5, it is in general difficult to compress the table while keeping the machine reversible.

# 4 Reversible cellular automata

Historically, Moore [27] and Myhill [41] first studied a problem having a relation to reversible cellular automata (RCAs). In particular, they studied the Garden-of-Eden problem that is closely related to injectivity and surjectivity of global functions of CAs. Since then, problems on injectivity and surjectivity have been studied more generally by many researchers [1, 24, 25, 45].

From the viewpoint of reversible computing RCAs are important, because they can be regarded as abstract spatio-temporal models of reversible physical space. It is especially interesting to see how complex functions such as computation-universality emerges from simple reversible local transition rules.

## 4.1 Definitions and basic properties

### 4.1.1 Cellular automata and reversibility

**Definition 4.1** *A* deterministic $k$-dimensional $m$-neighbor cellular automaton *(CA) is a system defined by*

$$A = (\mathbb{Z}^k, Q, (n_1, \cdots, n_m), f, \#),$$

*where $\mathbb{Z}$ is the set of all integers (hence $\mathbb{Z}^k$ is the set of all $k$-dimensional points with integer coordinates at which cells are placed), $Q$ is a non-empty finite set of states of each cell, $(n_1, \cdots, n_m)$ is an element of $(\mathbb{Z}^k)^m$ called a neighborhood ($m = 1, 2, \cdots$), $f : Q^m \to Q$ is a local function, and $\# \in Q$ is a quiescent state satisfying $f(\#, \cdots, \#) = \#$.*

*A* $k$-dimensional configuration *over the set $Q$ is a mapping $\alpha : \mathbb{Z}^k \to Q$. Let* $\mathrm{Conf}_k(Q)$ *denote the set of all $k$-dimensional configurations over $Q$, i.e.,* $\mathrm{Conf}_k(Q) = \{\alpha \mid \alpha : \mathbb{Z}^k \to Q\}$. *If $k$ is understood, we write it by* $\mathrm{Conf}(Q)$. *We say that a configuration $\alpha$ is* finite *iff the set $\{x \mid x \in \mathbb{Z}^k \wedge \alpha(x) \neq \#\}$ is finite. Otherwise, it is called* infinite.

*The global function $F : \mathrm{Conf}(Q) \to \mathrm{Conf}(Q)$ of $A$ is defined as the one that satisfies the following formula.*

$$\forall \alpha \in \mathrm{Conf}(Q),\ x \in \mathbb{Z}^k :\ F(\alpha)(x) = f(\alpha(x + n_1), \cdots, \alpha(x + n_m))$$

We then give notions of injectivity and invertibility of a CA.

**Definition 4.2** *Let $A = (\mathbb{Z}^k, Q, (n_1, \cdots, n_m), f, \#)$ be a CA. (1) $A$ is called an* injective CA *iff $F$ is one-to-one. (2) $A$ is called an* invertible CA *iff there is a CA $A' = (\mathbb{Z}^k, Q, N', f', \#)$ that satisfies the following condition.*

$$\forall \alpha, \beta \in \mathrm{Conf}(Q) :\ F(\alpha) = \beta \text{ iff } F'(\beta) = \alpha,$$

*where $F$ and $F'$ are the global functions of $A$ and $A'$, respectively.*

We can derive the following theorem from the results independently shown by Hedlund [12], and Richardson [45].

**Theorem 4.1** *(Hedlund [12] and Richardson [45]) Let $A$ be a CA. $A$ is injective iff it is invertible.*

By this theorem, we can see injectivity and invertibility in CAs are equivalent. Hereafter, we use the terminology "reversible CA" (RCA) rather than injective or invertible CA.

Note that in the case of reversible logic elements or RTMs, reversibility implies invertibility almost trivially from their definitions (where "invertibility" means there is an inverse logic element or an inverse RTM that undoes a computation by the original one). Therefore, the notion of reversibility can be directly defined for these models without introducing invertibility. On the other hand, since injectivity of a CA is defined on its global function, which is a mapping from configurations to configurations, it is not a trivial problem whether there is an inverse CA. Hence there was a need to introduce invertibility.

### 4.1.2 Block cellular automata and partitioned cellular automata

Since we want to find interesting RCAs here, we need design methods for RCAs. But, it is difficult to construct an RCA with a desired property. This is because the following reason. Kari [16] showed that it is undecidable whether a given two-dimensional CA is reversible or not (it also holds for the higher dimensional case). Therefore, we have to find some decidable subclass of RCAs. Otherwise, designing two-dimensional RCAs will be extremely difficult.

For the case of one-dimensional CA, Amoroso and Patt [2] showed that reversibility is decidable for one-dimensional CAs, and gave a decision algorithm. There are also several studies on enumerating all reversible one-dimensional CAs (e.g., [8, 28]). But, it is still impractical to use the decision algorithm for the design of RCAs, since it is not so easy to test reversibility of a given CA.

So far, several special frameworks are proposed for designing RCAs. They are, for example, CAs with block rules [23, 50], partitioned CAs [30], CAs with second order rules [23, 50].

Margolus [23] proposed a "block CA," which is often called a CA with Margolus neighborhood. In his original model, cells are grouped into blocks of size $2 \times 2$ as shown in Fig. 29. He proposed a specific set of local transformation rules, i.e., "block rules," shown in Fig. 30. This CA evolves as follows: At time 0 the local transformation is applied to every solid line block, then at time 1 to every dotted line block, and so on, alternatively. Since this transformation is one-to-one, we can uniquely retrieve the previous configuration provided that the parity of time is given. In this way, one can obtain reversible CAs, by giving one-to-one block rules. However, CAs with Margolus neighborhood are not conventional CAs, because each cell should know the relative position in a block and the parity of time besides its own state.

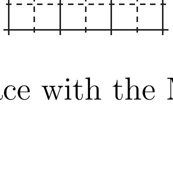


Figure 29: A cellular space with the Margolus neighborhood.

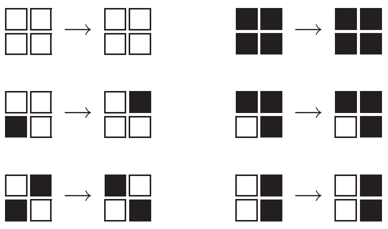


Figure 30: Block rules for the Margolus RCA [23]. (Rotation-symmetry is assumed here. Hence, rules obtained by rotating both sides of a rule are also included. )

Partitioned cellular automata (PCA) has some similarity to block CAs. But, resulting reversible CAs are in the framework of conventional CA (in other words, a PCA is a special subclass of a CA). In addition, flexibility of neighborhood is rather high. Shortcomings of PCA is that the number of states per cell becomes large.

**Definition 4.3** *A* deterministic $k$-dimensional $m$-neighbor partitioned cellular automaton *(PCA) is a system defined by*

$$P = (\mathbb{Z}^k, (Q_1, \cdots, Q_m), (n_1, \cdots, n_m), f, (\#, \cdots, \#)),$$

*where* $\mathbb{Z}$ *is the set of all integers,* $Q_i$ *(*$i = 1, \cdots, m$*) is a non-empty finite set of states of the i-th part of each cell (thus the state set of each cell is* $Q = Q_1 \times \cdots \times Q_m$*),* $(n_1, \cdots, n_m) \in (\mathbb{Z}^k)^m$ *is a neighborhood,* $f : Q \to Q$ *is a local function, and* $(\#, \cdots, \#) \in Q$ *is a quiescent state satisfying* $f(\#, \cdots, \#) = (\#, \cdots, \#)$*.*

*The notion of a (finite or infinite) configuration is defined similarly as in CA. Let* $p_i : Q \to Q_i$ *be the projection function such that* $p_i(q_1, \cdots, q_m) = q_i$ *for all* $(q_1, \cdots, q_m) \in Q$*. The global function* $F : \mathrm{Conf}(Q) \to \mathrm{Conf}(Q)$ *of* $P$ *is defined as the one that satisfies the following formula.*

$$\forall \alpha \in \mathrm{Conf}(Q), x \in \mathbb{Z}^k : F(\alpha)(x) = f(p_1(\alpha(x + n_1)), \cdots, p_m(\alpha(x + n_m)))$$

By the above definition, a one-dimensional radius 1 (3-neighbor) PCA $P_{\mathrm{1d}}$ can be defined as follows.

$$P_{\mathrm{1d}} \;=\; (\mathbb{Z}, (L, C, R), (1, 0, -1), f, (\#, \#, \#))$$

Each cell is divided into three parts, i.e., left, center, and right parts, and their state sets are $L, C$, and $R$. The next state of a cell is determined by the present states of the left part of the right-neighbor cell, the center part of this cell, and the right part of the left-neighbor cell (not depending on the whole three parts of the three cells). Fig. 31 shows its cellular space, and how the local function $f$ works.

Let $(l, c, r), (l', c', r') \in L \times C \times R$. If $f(l, c, r) = (l', c', r')$, then this equation is called a local rule (or simply a rule) of the PCA $P_{\mathrm{1d}}$, and it is sometimes written in a pictorial form as shown in Fig. 32. Note that, in the pictorial representation, the arguments of the lefthand side of $f(l, c, r) = (l', c', r')$ appear in a reverse order.

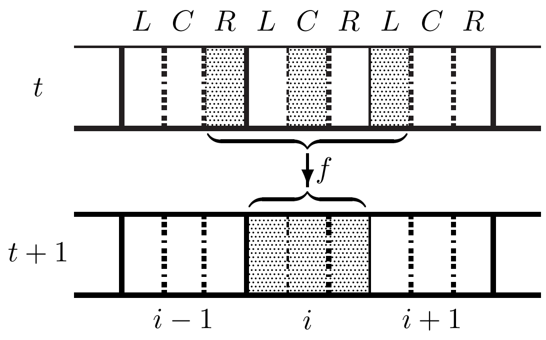


Figure 31: Cellular space of a one-dimensional 3-neighbor PCA $P_{\text{1d}}$, and its local function $f$.

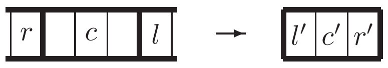


Figure 32: A pictorial representation of a local rule $f(l, c, r) = (l', c', r')$ of a one-dimensional 3-neighbor PCA $P_{\text{1d}}$.

A $k$-dimensional PCA is also defined similarly. Let

$$P = (\mathbb{Z}^k, (Q_1, \cdots, Q_m), (n_1, \cdots, n_m), f, (\#_1, \cdots, \#_m))$$

be a $k$-dimensional PCA, and $F$ be its global function. Then, it is easy to show the following proposition (a proof for the one-dimensional case given in [30] can be extended to higher dimensions).

**Proposition 4.1** *Let $P$ be a PCA, and $f$ and $F$ be its local and global functions. Then, $f$ is one-to-one iff $F$ is one-to-one.*

It is also easy to see that the class of PCAs is a subclass of CAs. More precisely, for any $k$-dimensional $m$-neighbor PCA $P$, we can obtain a $k$-dimensional $m$-neighbor CA $A$ whose global function is identical with that of $P$, by extending the domain of the local function of $P$. By above, if we want to construct an RCA, it is sufficient to give a PCA whose local function $f$ is one-to-one. This makes a design of an RCA feasible.

### 4.1.3 Simulating irreversible CAs by reversible ones

Toffoli [47] first showed that for every irreversible CA there exists a reversible one that simulates the former by increasing the dimension by one.

**Theorem 4.2** *(Toffoli [47]) For any $k$-dimensional (irreversible) CA $A$, we can construct a $k+1$-dimensional RCA $A'$ that simulates $A$.*

From this result, computation-universality of two-dimensional RCA is derived, since it is easy to embed a Turing machine in an (irreversible) one-dimensional CA. In the next section, we shall see this result is improved, i.e., one-dimensional RCA is computation-universal.

## 4.2 1-D universal reversible cellular automata

### 4.2.1 Simulating reversible Turing machines

It is possible to show computation-universality of one-dimensional (1-D) RCAs by constructing RCAs that can simulate reversible Turing machines directly.

It is shown in [29] that for any irreversible one-tape TM, there is a reversible one-tape two-symbol TM which simulates the former. In fact, to prove computation-universality of a 1-D PCA, it is convenient to simulate a reversible one-tape TM.

**Theorem 4.3** *(Morita and Harao [30]) For any reversible one-tape Turing machine $T$, there is a 1-D reversible PCA $P$ that simulates the former. Hence, 1-D RCA is computation-universal.*

In [30] $T$ is given in the quadruple form, but here we assume the quintuple form. Hence, the method of constructing $P$ is slightly modified. Let $T = (Q, S, q_0, q_f, s_0, \delta)$ be a reversible one-tape TM in the quintuple form. We assume that $q_0$ does not appear as the fifth element in any quintuple in $\delta$ (because we can always construct such a reversible TM from an irreversible one [29]). We can also assume there is no quintuple of the form $[p, s, t, 0, q]$ in $\delta$. A reversible PCA $P = (\mathbb{Z}, (L, C, R), (1, 0, -1), f, (\#, s_0, \#))$ that simulates $T$ is as follows. The state sets $L, C$, and $R$ are:

$$L = R = Q \cup \{\#\}, \quad C = S \cup \hat{S},$$

where $\hat{S} = \{\hat{s} \mid s \in S\}$. Let $Q_+ = \{q \mid \exists p \in Q\ \exists s,t \in S\,([p,s,t,+,q] \in \delta)\}$, $Q_- = \{q \mid \exists p \in Q\ \exists s,t \in S\,([p,s,t,-,q] \in \delta)\}$, and $Q_H = \{p \mid \neg(\exists q \in Q\ \exists s,t \in S\ \exists d \in \{-,+\}\,([p,s,t,d,q] \in \delta))\}$. Since $T$ is an RTM, $Q_+ \cap Q_- = \emptyset$. Also note that $q_f \in Q_H$. The local function $f$ is as below.

(1) For every $s, t \in S$, and $q \in Q - \{q_0\}$,

$$\begin{array}{rcll} f(\#, s, \#) &=& (\#, s, \#), & \\ f(\#, \hat{s}, \#) &=& (\#, \hat{s}, \#), & \\ f(\#, s, q_0) &=& (\#, s, q_0), & \\ f(\#, \hat{s}, q_0) &=& (\#, t, q) & \text{if } [q_0, s, t, +, q] \in \delta, \\ f(\#, \hat{s}, q_0) &=& (q, t, \#) & \text{if } [q_0, s, t, -, q] \in \delta, \end{array}$$

(2) For every $p \in Q - (Q_H \cup \{q_0\})$, $q \in Q - \{q_0\}$, and $s, t \in S$,

$$\begin{array}{ll} f(\#, s, p) = (\#, t, q) & \text{if } p \in Q_+ \text{ and } [p, s, t, +, q] \in \delta, \\ f(p, s, \#) = (\#, t, q) & \text{if } p \in Q_- \text{ and } [p, s, t, +, q] \in \delta, \\ f(\#, s, p) = (q, t, \#) & \text{if } p \in Q_+ \text{ and } [p, s, t, -, q] \in \delta, \\ f(p, s, \#) = (q, t, \#) & \text{if } p \in Q_- \text{ and } [p, s, t, -, q] \in \delta. \end{array}$$

(3) For every $q \in Q_H$ and $s \in S$,

$$\begin{array}{ll} f(\#, s, q) = (\#, \hat{s}, q) & \text{if } q \in Q_+, \\ f(q, s, \#) = (q, \hat{s}, \#) & \text{if } q \in Q_-. \end{array}$$

We can verify that the right-hand side of each rule differs from that of any other rule, because $T$ is deterministic and reversible. If the initial computational configuration of $T$ is

$$\cdots s_0 t_1 \cdots q_0 t_i \cdots t_n s_0 \cdots$$

then set $P$ to the following configuration.

$$\cdots, (\#, s_0, q_0), (\#, t_1, \#), \cdots, (\#, \hat{t_i}, \#), \cdots, (\#, t_n, \#), (\#, s_0, \#), \cdots$$

The simulation process starts when a right-moving signal $q_0$ meets the center state $\hat{t_i}$. It is easily seen that, from this configuration, $P$ can correctly simulates $T$ by the rules in (2). If $T$ becomes a halting state $q$ ($\in Q_H$), then the signal $q$ travels leftward or rightward changing each center state $s$ into $\hat{s}$. Note that $P$ itself cannot halt, because $P$ is reversible, but the final result is kept unchanged.

**Example 4.1** *Consider a reversible TM* $T_{\mathrm{mod}\,3} = (Q, \{b, 0, 1, 2\}, q_0, q_f, b, \delta)$*, where* $Q = \{q_0, \cdots, q_6, q_f\}$*, and* $\delta$ *is as below.*

$$\begin{array}{llll}
\delta = \{ & [\ q_0, b, b, +, q_3\ ], & & \\
& [\ q_1, b, b, +, q_4\ ], & [\ q_1, 0, 0, +, q_2\ ], & [\ q_1, 1, 1, +, q_3\ ], \\
& [\ q_2, b, b, +, q_5\ ], & [\ q_2, 0, 0, +, q_1\ ], & [\ q_2, 1, 1, +, q_2\ ], \\
& [\ q_3, b, b, +, q_6\ ], & [\ q_3, 0, 0, +, q_3\ ], & [\ q_3, 1, 1, +, q_1\ ], \\
& [\ q_4, b, 1, -, q_f\ ], & [\ q_5, b, 2, -, q_f\ ], & [\ q_6, b, 0, -, q_f\ ]\ \}.
\end{array}$$

*For a given binary number $n$ on the tape,* $T_{\mathrm{mod}\,3}$ *computes* ($n$ mod 3) *and writes it on the tape as in Fig. 33. A simulation process of* $T_{\mathrm{mod}\,3}$ *by a reversible PCA* $P_{\mathrm{mod}\,3}$*, which is constructed by the method described above, is in Fig. 34.*

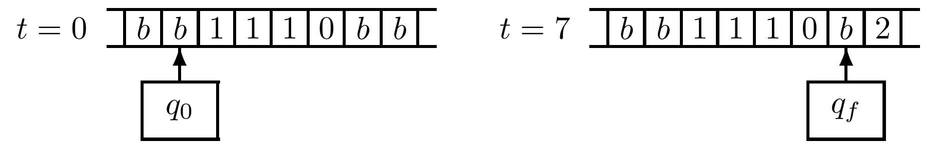


Figure 33: The initial and the final computational configuration of $T_{\mathrm{mod}\,3}$ for a given binary input 1110.

### 4.2.2 Simulating cyclic tag systems

From Theorems 3.3 and 4.3, we see there is a universal RCA in which any recursive function can be computed. But, if we use the construction method employed in Theorem 4.3, the number of states of the universal RCA becomes rather large, i.e., $16 \times 12 \times 16 = 3072$. However, simulating CTAGs by a reversible PCA directly, a relatively small universal RCA is obtained. In [40], a 36-state reversible PCA is given, but we show here an improved version.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $t=0$ |  | $b$ | $q_0$ |  | $\hat{b}$ |  |  | 1 |  |  | 1 |  |  | 1 |  |  | 0 |  |  | $b$ |  |  | $b$ |  |
| $t=1$ |  | $b$ |  |  | $b$ | $q_3$ |  | 1 |  |  | 1 |  |  | 1 |  |  | 0 |  |  | $b$ |  |  | $b$ |  |
| $t=2$ |  | $b$ |  |  | $b$ |  |  | 1 | $q_1$ |  | 1 |  |  | 1 |  |  | 0 |  |  | $b$ |  |  | $b$ |  |
| $t=3$ |  | $b$ |  |  | $b$ |  |  | 1 |  |  | 1 | $q_3$ |  | 1 |  |  | 0 |  |  | $b$ |  |  | $b$ |  |
| $t=4$ |  | $b$ |  |  | $b$ |  |  | 1 |  |  | 1 |  |  | 1 | $q_1$ |  | 0 |  |  | $b$ |  |  | $b$ |  |
| $t=5$ |  | $b$ |  |  | $b$ |  |  | 1 |  |  | 1 |  |  | 1 |  |  | 0 | $q_2$ |  | $b$ |  |  | $b$ |  |
| $t=6$ |  | $b$ |  |  | $b$ |  |  | 1 |  |  | 1 |  |  | 1 |  |  | 0 |  |  | $b$ | $q_5$ |  | $b$ |  |
| $t=7$ |  | $b$ |  |  | $b$ |  |  | 1 |  |  | 1 |  |  | 1 |  |  | 0 |  |  | $b$ |  | $q_f$ | 2 |  |
| $t=8$ |  | $b$ |  |  | $b$ |  |  | 1 |  |  | 1 |  |  | 1 |  |  | 0 |  | $q_f$ | $\hat{b}$ |  |  | 2 |  |
| $t=9$ |  | $b$ |  |  | $b$ |  |  | 1 |  |  | 1 |  |  | 1 |  | $q_f$ | $\hat{0}$ |  |  | $\hat{b}$ |  |  | 2 |  |
| $t=10$ |  | $b$ |  |  | $b$ |  |  | 1 |  |  | 1 |  | $q_f$ | $\hat{1}$ |  |  | $\hat{0}$ |  |  | $\hat{b}$ |  |  | 2 |  |
| $t=11$ |  | $b$ |  |  | $b$ |  |  | 1 |  | $q_f$ | $\hat{1}$ |  |  | $\hat{1}$ |  |  | $\hat{0}$ |  |  | $\hat{b}$ |  |  | 2 |  |
| $t=12$ |  | $b$ |  |  | $b$ |  | $q_f$ | $\hat{1}$ |  |  | $\hat{1}$ |  |  | $\hat{1}$ |  |  | $\hat{0}$ |  |  | $\hat{b}$ |  |  | 2 |  |
| $t=13$ |  | $b$ |  | $q_f$ | $\hat{b}$ |  |  | $\hat{1}$ |  |  | $\hat{1}$ |  |  | $\hat{1}$ |  |  | $\hat{0}$ |  |  | $\hat{b}$ |  |  | 2 |  |
| $t=14$ | $q_f$ | $\hat{b}$ |  |  | $\hat{b}$ |  |  | $\hat{1}$ |  |  | $\hat{1}$ |  |  | $\hat{1}$ |  |  | $\hat{0}$ |  |  | $\hat{b}$ |  |  | 2 |  |

Figure 34: Simulating $T_{\text{mod}\,3}$ by a 1-D reversible PCA $P_{\text{mod}\,3}$. The state # is indicated by a blank.

**Theorem 4.4** *There is a 30-state 1-D reversible PCA $P_{30}$ that can simulate any CTAG on infinite (leftward-periodic) configurations.*

The PCA $P_{30}$ that has 30 states and operates on an infinite (but leftward- and rightward-periodic) configuration.

$$P_{30} = (\mathbf{Z}, (\{\#\}, \{\#, Y, N, +, -, {}^{\bullet}\}, \{\#, y, n, +, -\}), g_{30}, (\#, \#, \#)).$$

The state set of each cell is therefore $L \times C \times R$, and thus it has 30 states. The local function $g_{30} : L \times C \times R \to L \times C \times R$ is defined as follows.

$$\begin{array}{rcll}
g_{30}(\#,\#,\#) &=& (\#,\#,\#) & \\
g_{30}(\#,Y,\#) &=& (\#,\#,+) & \\
g_{30}(\#,N,\#) &=& (\#,\#,-) & \\
g_{30}(\#,c,r) &=& (\#,c,r) & \text{for } c \in \{Y, N, {}^{\bullet}\},\ \text{and } r \in \{y, n, +, -\} \\
g_{30}(\#,c,r) &=& (\#,c,r) & \text{for } c \in \{\#, -\},\ \text{and } r \in \{y, n\} \\
g_{30}(\#,c,r) &=& (\#,r,c) & \text{for } c, r \in \{+, -\} \\
g_{30}(\#,+,y) &=& (\#,Y,\#) & \\
g_{30}(\#,+,n) &=& (\#,N,\#) & \\
g_{30}(\#,{}^{\bullet},\#) &=& (\#,+,n) &
\end{array}$$

Note that $P_{30}$ is in fact a two neighbor PCA, since the state set of the left part is $L = \{\#\}$. Since there is no pair of distinct local rules whose right-hand sides are the same, $P_{30}$ is an RCA.

Consider the following example of a CTAG $C_2 = (3, \{Y, N\}, (\text{halt}, NY, NN))$. If we give $NYN$ to $C_2$ as an initial string, then

$$(NYN, 0) \Rightarrow (YN, 1) \Rightarrow (NNY, 2) \Rightarrow (NY, 0) \Rightarrow (Y, 1) \Rightarrow \cdots$$

is a computation starting from $NYN$.

The initial configuration of $P_{30}$ for $C_2$ is set as follows (see the first row of Fig. 35). The initial string $NYN$ is given in the center parts of some three consecutive cells. Production rules are expressed by a sequence of the right-part states $y, n$, and # in a reverse order, where # is a delimiter indicating the beginning of a rule. Hence, one cycle of the rules $(\text{halt}, NY, NN)$ is represented by the sequence *nn#yn##*, where "halt" is expressed by an empty string. Since the rules $(\text{halt}, NY, NN)$ are applied cyclically, infinite copies of the sequence *nn#yn##* should be given in the initial configuration of $P_{30}$.

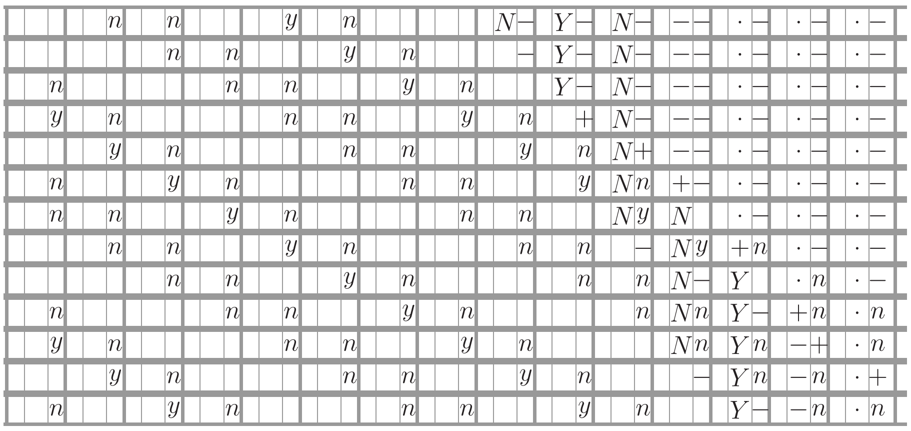


Figure 35: Simulating the CTAG $C_2$ by the reversible PCA $P_{30}$.

A rewriting process of a CTAG is simulated as in Fig. 35. If the signal # meets the state $Y$ (or $N$, respectively), which is the head symbol of a rewritten string, then it changes $Y$ ($N$) to the state #, and the signal itself becomes $+$ ($-$). This signal $+$ ($-$, respectively) goes rightward through the string consisting of $Y$'s and $N$'s. If it meets the center state $+$ or $-$, then the latter is replaced by $+$ ($-$) indicating the head symbol was $Y$ ($N$). Signals $y$ and $n$ travel rightward until it meets $+$ or $-$. If $y$ ($n$, respectively) meets $+$, then the signal becomes $Y$ ($N$) and is fixed at this position, and $+$ is shifted to the right by one cell. If $y$ or $n$ meets $-$, then the signal simply continues to travel rightward without being fixed. By this, the rewriting process is simulated.

Note that the reversible PCA $P_{30}$ does not deal with halting of a CTAG. Since the result of a computing will be destroyed after the CTAG halts, the evolution of the $P_{30}$ should be always watched from the outside whether the CTAG halts. Of course, it can be handled inside of the PCA by increasing the number of states. The following result shows there is a 98-state reversible PCA that can deal with halting of a CTAG, and operates on finite configurations.

**Theorem 4.5** *(Morita [40]) There is a 98-state 1-D reversible PCA $P_{98}$ that can simulate any CTAG on finite configurations. It also deals with halting of a CTAG, i.e., it keeps the final result of the CTAG undestroyed.*

## 4.3 2-D universal reversible cellular automata

As it is shown in section 4.2, even in the 1-D case, there are universal RCAs with relatively small number of states. On the other hand, in the 2-D case, several universal RCAs with much simpler local functions exist.

### 4.3.1 Simulating a Fredkin gate

Here, we discuss several models of 2-D RCAs in which a Fredkin gate (as well as signal routing mechanisms) can be embedded. By Theorem 2.5, any RSM can be embedded in such RCAs as a completely garbageless circuit. Hence, any RTM (including a URTM) can be realized as an infinite configuration in them.

Margolus [23] first proposed such a type of an RCA. It was shown that in his RCA with the block rules given in Fig. 30 the BBM can be realized. Fig. 36 shows a reflection of a ball by a mirror, where a ball is represented by two cells in black states. Collision of two balls is also simulated in this cellular space. Hence, the Fredkin gate can be constructed by the method discussed in section 2.3.1.

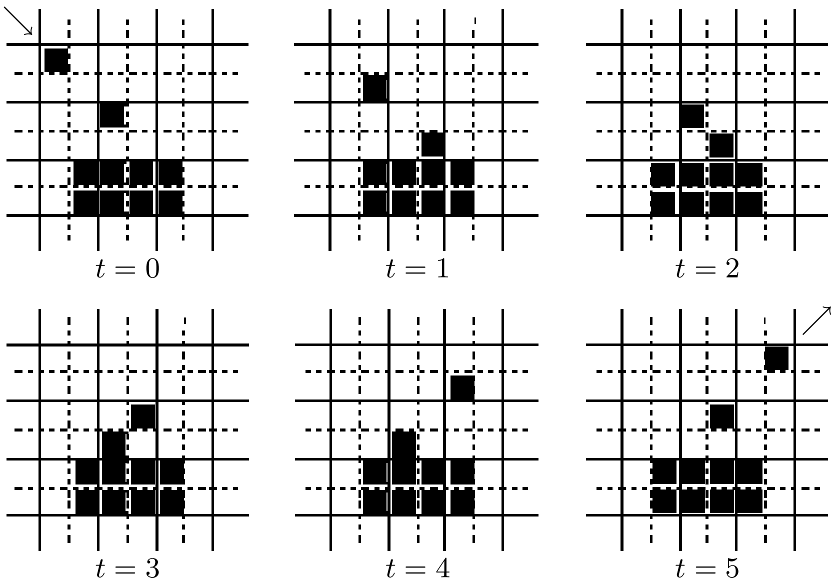


Figure 36: Reflection of a ball by a reflector in the Margolus' RCA [23].

Morita and Ueno [32] used the framework of PCA, and gave two models $S_1$ and $S_2$ of $2^4$-state RCAs in which a Fredkin gate can be realized. Here we show the second model $S_2$ that has the set of transition rules shown in Fig. 37. It is rotation-symmetric but not reflection-symmetric. Fig. 38 shows a configuration of a switch gate, where each ball is represented by two black dots.

Imai and Morita [14] proposed a $2^3$-state reversible PCA model $T_1$ on the triangular lattice. It is rotation-symmetric but not reflection-symmetric. Its local function is extremely simple as shown in Fig. 39. Signal routing, crossing, and delay are very complex to realize, because a kind of "wall" is necessary to make a signal go straight ahead. Thus the size of the configuration of a Fredkin gate is very large ($26 \times 220$) as in Fig. 40.

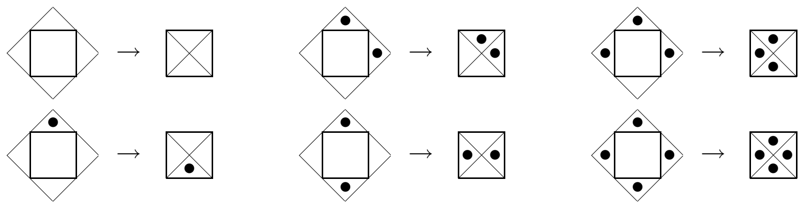


Figure 37: The local rules of the 16-state rotation-symmetric reversible PCA $S_2$ [32].

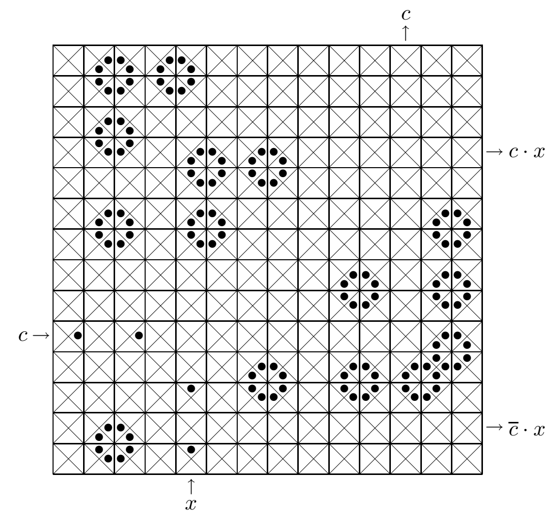


Figure 38: A switch gate realized in the reversible PCA $S_1$ [32].

We summarize the above as follows.

**Theorem 4.6** *Any RTM can be simulated in any of the following 2-D RCAs with infinite configurations.*
*1. The 2-state RCA with Margolus neighborhood (Margolus [23]).*
*2. The 16-state reversible PCAs $S_1$ and $S_2$ (Morita and Ueno [32]).*
*3. The 8-state reversible triangular PCA $T_1$ (Imai and Morita [14]).*

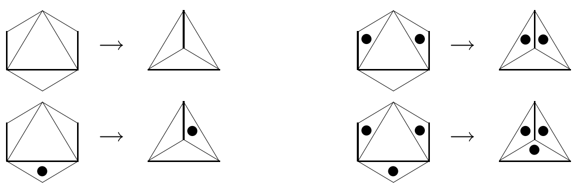


Figure 39: The local rules of the 8-state rotation-symmetric reversible triangular PCA $T_1$ [14].

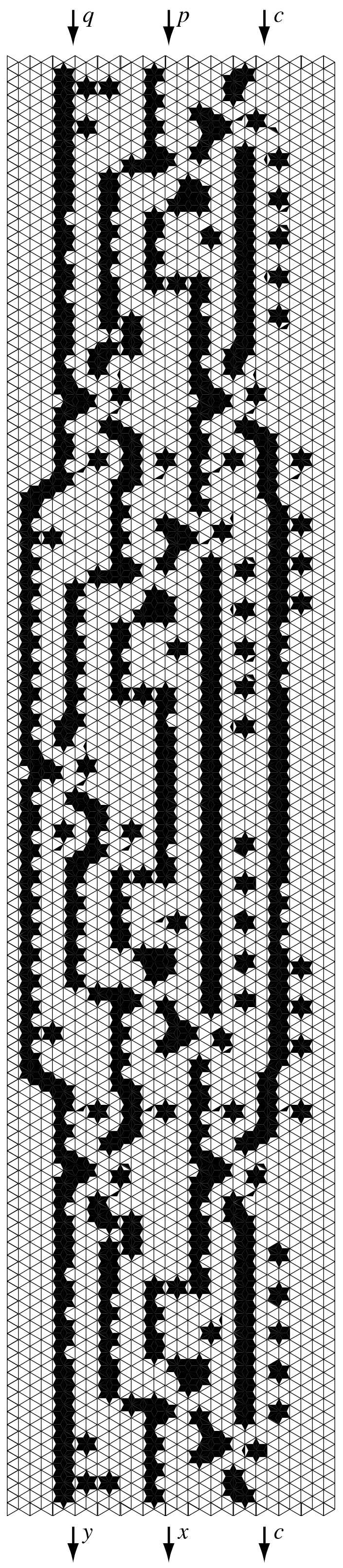


Figure 40: A Fredkin gate realized in the 8-state reversible triangular PCA $T_1$ [14].

### 4.3.2 Simulating the rotary element and reversible counter machines

If we try to design an RCA in which a reversible computer can be realized as a finite and compact configuration, it is convenient to use a reversible counter machine (RCM) rather than an RTM. As we shall see in this section, an RCM can be implemented simply in a cellular space by using REs and some additional elements as reversible devices of an intermediate level. A *counter machine* is a system made of a finite-state control and counters each of which can store a non-negative integer. It can be formalized as a kind of multi-tape Turing machine with read-only heads, where each tape has the symbol $Z$ (it stands for "zero") in the leftmost square, and the symbol $P$ ("positive") in all other squares (Fig. 41). A reversible CM (RCM) is defined in a similar manner as in RTM, (for its precise definition see [33]).

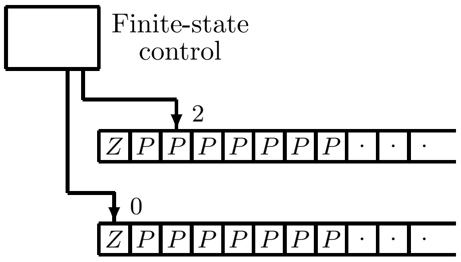


Figure 41: A counter machine with two counters.

It is known that a CM with two counters is computation-universal [26]. This result also holds even if the reversibility constraint is added.

**Theorem 4.7** *(Morita [33]) For any Turing machine $T$, there is a deterministic reversible CM with two counters $M$ that simulates $T$.*

In [36], a model of a $3^4$-state reversible PCA $P_3$ is shown, in which any RCM can be embedded.

**Theorem 4.8** *(Morita, Tojima, Imai and Ogiro [36]) There is a 81-state reversible PCA $P_3$ with finite configurations in which any reversible CM with two counters can be simulated.*

$P_3$ has the local transition rules shown in Fig. 42. In $P_3$, five kinds of signal processing elements shown in Fig. 43 can be realized as intermediate level reversible devices. An LR-turn element, an R-turn element, and a reflector are used for signal routing. A position marker is a device to keep a head position of a CM, and realized by a single ○, which rotates clockwise at a certain position by the rule (a) in Fig. 42, and can be pushed or pulled by one cell by giving an appropriate signal.

Fig. 44 shows an example of a configuration for a 2-counter RCM in $P_3$. The left half of it is the finite-state control of the RCM, while the right half simulates two counters. Since head positions of an RCM is represented by position markers, the whole configuration is

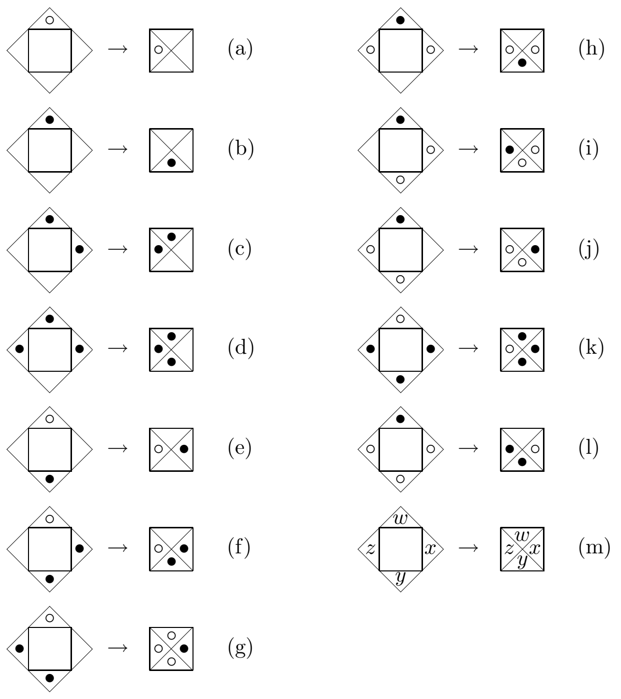


Figure 42: The local rules of the $3^4$-state rotation-symmetric reversible PCA $P_3$ [36]. The rule scheme (m) represents 33 rules not specified by (a)–(l), where $w, x, y, z \in \{\text{blank}, \circ, \bullet\}$.

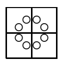
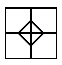
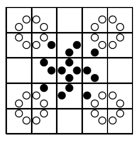
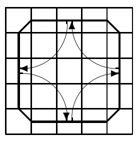
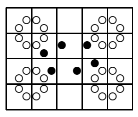
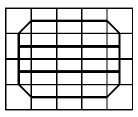
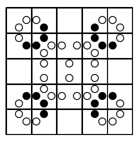
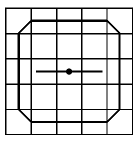
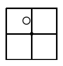
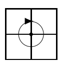
| Element Name | Pattern | Symbolic Notation |
|---|---|---|
| LR-turn element |  |  |
| R-turn element |  |  |
| Reflector |  |  |
| Rotary element |  |  |
| Position Marker |  |  |

Figure 43: Basic elements realized in the reversible PCA $P_3$.

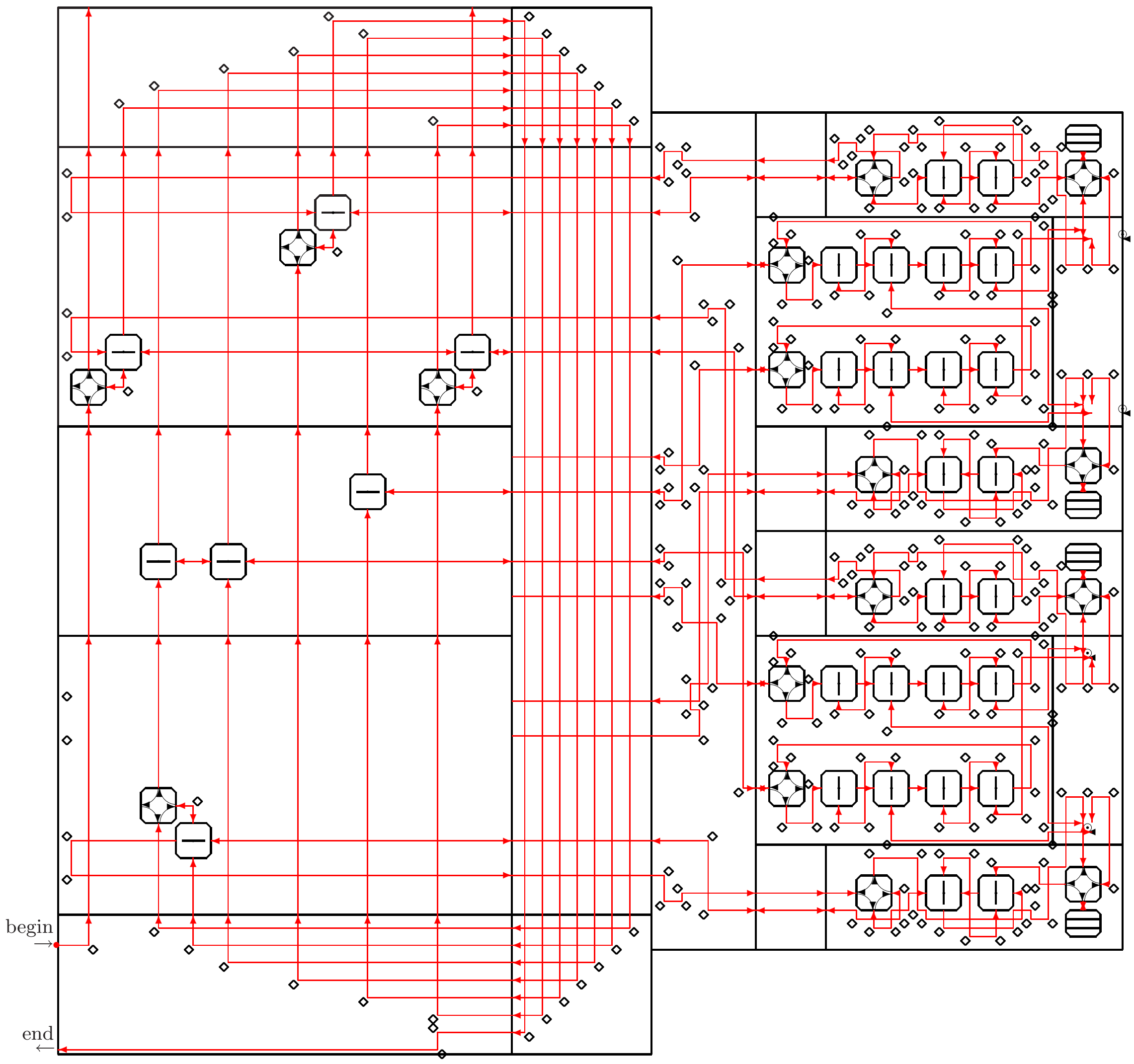


Figure 44: A configuration (represented by a diagram notation) of an example of an RCM, which computes the function $2x + 2$, in the reversible PCA $P_3$.

finite. In this construction, computation is carried out by a single signal that interacts with REs and position markers. Setting the position markers appropriately, and giving a single black dot to the "begin" port, it starts to compute. If the computation terminates, then the signal goes out from the "end" port leaving the answer in some counter.

## 4.4 Other issues on reversible cellular automata

There are still many other topics on RCAs not dealt with in the previous sections. In fact, RCAs have rich abilities of information processing besides the ability of computing recursive functions. For example, RCAs can solve the firing squad synchronization problem (FSSP). Imai and Morita [13] showed a $3n$-step solution by using 99-state reversible PCA for synchronizing $n$ cells. Self-reproduction is also possible in RCAs. There are a 2-D reversible PCA [34] and even 3-D reversible PCA [15] in which various shapes of objects can self-reproduce. Finally, we note that the papers [17, 50] are good surveys on RCAs and CAs with different viewpoints.

## 5 Concluding remarks

In this paper, we investigated and discussed how computing can be performed in reversible systems. We used several models of reversible computing chosen from variety of levels: reversible Turing machines, reversible cellular automata, reversible logic elements, reversible physical models, and a few others. Though these models have different features, they are closely related each other: some models are embeddable in other models relatively simply. In particular, any reversible model of a macroscopic level such as a reversible Turing machine can be implemented in a model of microscopic level like the Billiard Ball Model as a closed system.

In reversible systems, computation can be often carried out in a very different manner from conventional computer. For example, reversible Turing machines can be constructed very simply by using only rotary elements, where classical notions of gates like AND, OR, and NOT are not explicitly used at all. Such a fact suggests that there will be still other novel and unique methods of computing in reversible systems.

Since this paper is not an exhaustive survey on reversible computing, many important topic are missing: e.g., relations to thermodynamics of computing [5, 6], and to quantum computing [10]. We also omitted a topic on hardware realization of reversible computing systems, because we discussed them from the standpoint of the computation theory. But, it is an interesting and challenging problem, though very hard to solve.

## References

[1] Amoroso, S. and Cooper, G., The Garden of Eden theorem for finite configurations, *Proc. Amer. Math. Soc.*, **26**, 158–164 (1970).

[2] Amoroso, S. and Patt, Y.N., Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures, *J. Comput. Syst. Sci.*, **6**, 448–464 (1972).

[3] Baiocchi, C., Three small universal Turing machines, *3rd Int. Conference on Machines, Computations, and Universality*, LNCS 2055, Springer-Verlag, 1–10 (2001).

[4] Bennett, C.H., Logical reversibility of computation, *IBM J. Res. Dev.*, **17**, 525–532 (1973).

[5] Bennett, C.H., The thermodynamics of computation, *Int. J. Theoret. Phys.*, **21**, 905–940 (1982).

[6] Bennett, C.H., Notes on the history of reversible computation, *IBM J. Res. Dev.*, **32**, 16–23 (1988).

[7] Bennett, C.H., Time/space trade-offs for reversible computation, *SIAM J. Comput.*, **18**, 766–776 (1989).

[8] Boykett, T., Efficient exhaustive listings of reversible one dimensional cellular automata, *Theoret. Comput. Sci.*, **325**, 215–247 (2004).

[9] Cook, M., Universality in elementary cellular automata, *Complex Systems*, **15**, 1–40 (2004).

[10] Feynman, R.P., *Feynman lectures on computation* (eds., A.J.G. Hey and R.W. Allen), Perseus Books, Reading, Massachusetts (1996).

[11] Fredkin, E., and Toffoli, T., Conservative logic, *Int. J. Theoret. Phys.*, **21**, 219–253 (1982).

[12] Hedlund, G.A., Endomorphisms and automorphisms of the shift dynamical system, *Math. Systems Theory*, **3**, 320-375 (1969).

[13] Imai, K., and Morita, K., Firing squad synchronization problem in reversible cellular automata, *Theoret. Comput. Sci.*, **165**, 475–482 (1996).

[14] Imai, K., and Morita, K., A computation-universal two-dimensional 8-state triangular reversible cellular automaton, *Theoret. Comput. Sci.*, **231**, 181–191 (2000).

[15] Imai, K., Hori, T., and Morita, K., Self-reproduction in three-dimensional reversible cellular space, *Artificial Life*, **8**, 155–174 (2002).

[16] Kari, J., Reversibility and surjectivity problems of cellular automata, *J. Comput. Syst. Sci.*, **48**, 149–182 (1994).

[17] Kari, J., Theory of cellular automata: A survey, *Theoret. Comput. Sci.*, **334**, 3–33 (2005).

[18] Kudlek, M., and Rogozhin, Y., A universal Turing machine with 3 states and 9 symbols, *Developments in Language Theory (DLT 2001)*, LNCS 2295, Springer-Verlag, 311–318 (2002).

[19] Landauer, R., Irreversibility and heat generation in the computing process, *IBM J. Res. Dev.*, **5**, 183–191 (1961).

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

[21] Lee, J., Peper, F., Adachi, S. and Morita, K., An asynchronous cellular automaton implementing 2-state 2-input 2-output reversed-twin reversible elements, *Proc. Eighth Int. Conf. on Cellular Automata for Research and Industry* (ACRI 2008), Yokohama (to appear).

[22] Margenstern, M., Surprising areas in the quest for small universal devices, MFCSIT, Cork, *Electronic Notes in Theoret. Comput. Sci.* (to appear).

[23] Margolus, N., Physics-like model of computation, *Physica*, **10D**, 81–95 (1984).

[24] Maruoka, A. and Kimura, M., Condition for injectivity of global maps for tessellation automata, *Inf. Control*, **32**,158–162 (1976).

[25] Maruoka, A. and Kimura, M., Injectivity and surjectivity of parallel maps for cellular automata, *J. Comput. Syst. Sci.*, **18**, 47–64 (1979).

[26] Minsky, M.L., *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, NJ (1967).

[27] Moore, E.F., Machine models of self-reproduction, in *Essays on Cellular Automata* (ed. A.W.Burks), University of Illinois Press, Urbana, pp.187–203 (1970).

[28] Mora, J.C.S.T, Vergara, S.V.C, Martinez, G.J., McIntosh, H.V., Procedures for calculating reversible one-dimensional cellular automata, *Physica D*, **202**, 134–141 (2005).

[29] Morita, K., Shirasaki, A., and Gono, Y., A 1-tape 2-symbol reversible Turing machine, *Trans. IEICE Japan*, **E-72**, 223–228 (1989).

[30] Morita, K., and Harao, M., Computation universality of one-dimensional reversible (injective) cellular automata, *Trans. IEICE Japan*, **E-72**, 758–762 (1989).

[31] Morita, K., A simple construction method of a reversible finite automaton out of Fredkin gates, and its related problem, *Trans. IEICE Japan*, **E-73**, 978–984 (1990).

[32] Morita, K., and Ueno, S., Computation-universal models of two-dimensional 16-state reversible cellular automata, *IEICE Trans. Inf. & Syst.*, **E75-D**, 141–147 (1992).

[33] Morita, K., Universality of a reversible two-counter machine, *Theoret. Comput. Sci.*, **168**, 303–320 (1996).

[34] Morita, K., and Imai, K., Self-reproduction in a reversible cellular space, *Theoret. Comput. Sci.*, **168**, 337–366 (1996).

[35] Morita, K., A simple reversible logic element and cellular automata for reversible computing, in *Proc. 3rd Int. Conf. on Machines, Computations, and Universality*, LNCS-2055, Springer-Verlag, 102–113 (2001).

[36] Morita, K., Tojima, Y., Imai, K., and Ogiro, T., Universal computing in reversible and number-conserving two-dimensional cellular spaces, in *Collision-based Computing* (ed. A. Adamatzky), 161–199, Springer-Verlag (2002).

[37] Morita, K., A new universal logic element for reversible computing, *Grammars and Automata for String Processing* (C. Martin-Vide, and V. Mitrana, eds.), Taylor & Francis, London, 285–294 (2003).

[38] Morita, K., Ogiro, T., Tanaka, K., and Kato, H., Classification and universality of reversible logic elements with one-bit memory, *Proc. 4th Int. Conf. on Machines, Computations, and Universality*, LNCS-3354, Springer-Verlag, 245–256 (2005).

[39] Morita, K., and Yamaguchi, Y., A universal reversible Turing machine, *Proc. 5th Int. Conf. on Machines, Computations, and Universality*, LNCS-4664, Springer-Verlag, 90–98 (2007).

[40] Morita, K., Simple universal one-dimensional reversible cellular automata, *J. Cellular Automata*, **2**, 159–165 (2007).

[41] Myhill, J., The converse of Moore's Garden-of-Eden theorem, in *Essays on Cellular Automata* (ed. A.W. Burks), University of Illinois Press, Urbana, 204–205 (1970).

[42] Neary, T., and Woods, D., Four small universal Turing machines, *Proc. 5th Int. Conf. on Machines, Computations, and Universality*, LNCS-4664, Springer-Verlag, 242–254 (2007).

[43] Ogiro, T., Kanno, A., Tanaka, K., Kato, H.. and Morita, K., Nondegenerate 2-state 3-symbol reversible logic elements are all universal, *Int. Journ. of Unconventional Computing*, **1**, 47–67 (2005).

[44] Petri, C.A., Grundsätzliches zur Beschreibung diskreter Prozesse, *Proc. 3rd Colloquium über Automatentheorie*, Birkhäuser Verlag, 121–140 (1967).

[45] Richardson, D., Tessellations with local transformations, *J. Comput. Syst. Sci.*, **6**, 373–388 (1972).

[46] Rogozhin, Y., Small universal Turing machines, *Theoretical Computer Science*, **168**, 215–240 (1996).

[47] Toffoli, T., Computation and construction universality of reversible cellular automata, *J. Comput. Syst. Sci.*, **15**, 213–231 (1977).

[48] Toffoli, T., Reversible computing, *Automata, Languages and Programming* (eds. J.W. de Bakker and J. van Leeuwen), LNCS-85, Springer-Verlag, 632–644 (1980).

[49] Toffoli, T., Bicontinuous extensions of invertible combinatorial functions, *Mathematical Systems Theory*, **14**, 12–23 (1981).

[50] Toffoli, T., and Margolus, N., Invertible cellular automata: a review, *Physica D*, **45**, 229–253 (1990).

[51] Woods, D., and Neary, T., Small semi-weakly universal Turing machines, *Proc. 5th Int. Conf. on Machines, Computations, and Universality*, LNCS-4664, Springer-Verlag, 303–315 (2007).