Reversible World of Cellular Automata
— Data set for the Golly simulator, and solutions to selected exercises —

Kenichi Morita*

May 2024

1 How to use the file "Reversible_World_of_CAs.zip"

The file "Reversible_World_of_CAs.zip" contains rule files and pattern files that are executable on the general purpose cellular automaton simulator *Golly* [10]. These files are for simulating various reversible cellular automata (RCAs), and other reversible computing systems given in the following book *Reversible World of Cellular Automata*. The files will greatly help readers to understand the contents of the book. The zip file also contains the file "_Solutions_for_selected_exercises.pdf" that describes solutions for exercise problems given in the book.

> K. Morita: *Reversible World of Cellular Automata*
> — *Fantastic Phenomena and Computing in Artificial Reversible Universe*,
> World Scientific Publishing, Singapore (2024).
> https://doi.org/10.1142/13516

Golly [10] is an excellent cellular automaton simulator developed by A. Trevorrow, T. Rokicki, T. Hutton *et al.* It can deal with very large patterns of cellular automata, and its simulation speed is quite fast. One can easily see fantastic evolution processes of RCAs and related systems by the following procedure.

1. Download the Golly system from https://golly.sourceforge.io/
2. Install the system on your computer.
3. Put the file "Reversible_World_of_CAs.zip" in the "Patters" folder of Golly.
4. Start the Golly simulator.
5. Select the zip file in the "Patterns" folder from Golly, and access any pattern file (a file with .rle or .mc) in the zip file.

Note that readers can know the usage of Golly by accessing its help menu.

Files contained in "Reversible_World_of_CAs.zip"

Many files are contained in the zip file "Reversible_World_of_CAs.zip". Files having .rle or .mc are *pattern files* for simulating CAs and other computing systems. They describe initial configurations of CAs and other systems. Files having .rule are *rule files*. They describe local functions of CAs or rules for simulating other systems. The rule files are automatically installed by Golly when users have selected the zip file.

*Currently Professor Emeritus of Hiroshima University, morita.rcomp@gmail.com

There are 96 pattern files in total. In them, 78 files have the file names that begin with "Ch_", while 18 files have the names that begin with "Cx_Exercise_". The former ones are for simulating CAs and other systems explained in the text of the book *Reversible World of Cellular Automata*. The latter ones are given as solutions of the exercise problems in the book. A file name of a pattern file contains chapter and section numbers, as well as a short title of the pattern, after the prefix "Ch_" or "Cx_Exercise_". A more detailed descriptions of the 96 pattern files are given in Section 2 below.

2 List of pattern files

Short explanations for the 96 pattern files contained in the zip file are given below. These files are for simulating reversible square partitioned cellular automata (SPCAs), reversible triangular partitioned cellular automata (TPCAs), reversible Turing machines (RTMs), reversible counter machines (RCMs), and circuits composed of reversible logic elements with memory (RLEMs). Although Golly is a CA simulator, it is also used for simulating computing models other than CAs. In particular, it is very useful to simulate very large RLEM-circuits, as well as very large configurations of CAs, for a huge number of steps.

The above computing models are explained in Section 3. However, readers are recommended to run the pattern files on Golly first, and observe their evolution processes. By this, one can roughly understand how the computing models work even if he/she does not read Section 3.

Note that some of these patterns require millions (or even billions) of steps to obtain results. Thus the simulation speed of Golly must be accelerated by pressing the "+" key (slow-down is by the "−" key). Recommended speed (*e.g.*, step $= 8^m - 8^n$, where 8^m and 8^n are the first and the last speed) can be indicated by pressing the button " **i** " (Show pattern information) of Golly.

Cover Page

- Ch_0__Cover_picture_generated_by_ESPCA-0925bf.rle
  Generates fascinating patterns like the cover picture of the book by the reversible ESPCA-0925bf.

Chapter 1. Cellular Automaton as an Artificial Digital World

- Ch_1_1_(1)_Fredkin_CA_CheshireCat.rle
  Shows the Fredkin's CA in which any initial pattern self-replicates.
- Ch_1_1_(2)_Game_of_Life_CheshireCat.rle
  Shows the evolution process of the Cheshire Cat pattern in the Game of Life (GoL).

Chapter 2. Elementary Partitioned Cellular Automata

- Ch_2_1_(1)_ESPCA-09458f_three_kinds_of_confs.rle
  Gives pre-periodic, pre-space-moving, and diameter-growing patterns in the irreversible ESPCA-09458f.
- Ch_2_1_(2)_ESPCA-0945df_three_kinds_of_confs.rle
  Gives periodic, space-moving, and diameter-growing patterns in the reversible ESPCA-0945df.
- Ch_2_2_(1)_ETPCA-0347_three_kinds_of_confs.rle
  Gives periodic, space-moving, and diameter-growing patterns in the reversible ETPCA-0347.
- Ch_2_2_(2)_ETPCA-0340_three_kinds_of_confs.rle
  Gives pre-periodic, pre-space-moving, and diameter-growing patterns in the irreversible ETPCA-0340.
- Cx_Exercise_2_4_ESPCA-0945df.rle
  Shows evolution processes of two patterns, which are periodic and diameter-growing.
- Cx_Exercise_2_5_ETPCA_0347.rle
  Shows evolution processes of two patterns, which are periodic and diameter-growing.

Chapter 3. Time-Reversal Symmetry in Reversible PCA

- Ch_3_2_(1)_ESPCA-01caef_T-symmetry.rle
  Gives a diagram that shows T-symmetry of ESPCA-01caef under the involution $H^{\mathrm{rev}} \circ H^{\mathrm{refl}}$.
- Ch_3_3_(1)_ETPCA-0527_T-symmetry.rle
  Shows a T-symmetric evolution in ETPCA-0527.
- Ch_3_3_(2)_ETPCA-0347_T-symmetry.rle
  Gives a diagram that shows T-symmetry of ETPCA-0347 under $H^{\mathrm{rev}} \circ H^{\mathrm{refl}}$.
- Ch_3_3_(3)_ETPCA-0347_Implosion_and_explosion.rle
  Shows a T-symmetric evolution in ETPCA-0347. We can see that explosions occur in both positive and negative time directions.
- Cx_Exercise_3_2_ESPCA-073a2f_T-symmetry.rle
  Shows a T-symmetric evolution in ESPCA-073a2f.
- Cx_Exercise_3_2_ESPCA-08cadf_Hrev.rle
  Shows that ESPCA-08cadf is used to perform the involution H^{rev}.
- Cx_Exercise_3_3_ESPCA-07ca2f_T-symmetry.rle
  Shows a T-symmetric evolution in ESPCA-07ca2f.

Chapter 4. Universal Systems of Reversible Computing

- Ch_4_1_(1)_RTM_examples.rle
  Gives three examples of RTMs: T_{parity}, T_{power} and T_{square}.
- Ch_4_1_(2)_RTM_prime.rle
  Gives an example of an RTM: T_{prime}.
- Ch_4_2_(1)_RCM_examples.rle
  Gives examples of RCMs in the quadruple form: M_{twice} and M_{exp}, and examples of RCMs in the program form: M_{move}, M_{twice} and M_{exp}.
- Ch_4_3_(1)_RSM_by_RE_and_RLEM_4-31.rle
  Shows how to compose a reversible sequential machine (RSM) out of rotary elements (REs) and RLEM 4-31.
- Ch_4_3_(2)_RE_by_RLEMs_2-3_and_2-4.rle
  Shows how to compose an RE by RLEMs 2-3 and 2-4.
- Ch_4_3_(3)_RTM_parity_by_RE.rle
  Shows how to compose an RTM T_{parity} out of REs, and how it is simulated.
- Ch_4_3_(4)_RTM_power_by_RE.rle
  Shows how to compose an RTM T_{power} out of REs, and how it is simulated.
- Ch_4_3_(5)_RTM_square_by_RE.rle
  Shows how to compose an RTM T_{square} out of REs, and how it is simulated.
- Ch_4_3_(6)_RTM_prime_by_RE.rle
  Shows how to compose an RTM T_{prime} out of REs, and how it is simulated.
- Ch_4_3_(7)_RTM_parity_by_RLEM_4-31.rle
  Shows how to compose an RTM T_{parity} out of RLEM 4-31, and how it is simulated.
- Ch_4_3_(8)_RTM_power_by_RLEM_4-31.rle
  Shows how to compose an RTM T_{power} out of RLEM 4-31, and how it is simulated.
- Cx_Exercise_4_08_RTMs.rle
  Gives a design of two RTMs T_{add} and T_{mult}.
- Cx_Exercise_4_09_RCMs.rle
  Gives a design of RCM M_{half} in the quadruple form.
- Cx_Exercise_4_10_RLEMs.rle
  Gives composing methods of several RLEMs by other RLEMs, and composing methods of RTMs T_{add} and T_{mult} out of REs.

Chapter 5. Fantastic Phenomena in Reversible PCAs

- Ch_5_2_1_1_(1)_ESPCA-01c5ef_periodic.rle
  Gives examples of periodic patterns in ESPCA-01c5ef.
- Ch_5_2_1_2_(1)_ESPCA-01753f_periodic.rle
  Gives examples of periodic patterns in ESPCA-01753f.
- Ch_5_2_1_2_(2)_ESPCA-0945df_periodic.rle
  Gives examples of periodic patterns in ESPCA-0945df.
- Ch_5_2_2_1_(1)_ETPCA-0157_periodic.rle
  Gives examples of periodic patterns in ETPCA-0157.
- Ch_5_2_2_1_(2)_ETPCA-0137_periodic.rle
  Gives examples of periodic patterns in ETPCA-0137.
- Ch_5_2_2_2_(1)_ETPCA-0347_periodicr.rle
  Gives examples of periodic patterns in ETPCA-0347.
- Ch_5_3_1_1_(1)_ESPCA-01caef_space-moving.rle
  Gives examples of space-moving patterns in ESPCA-01caef.
- Ch_5_3_1_1_(2)_ESPCA-016a7f_space-moving.rle
  Gives examples of space-moving patterns in ESPCA-016a7f.
- Ch_5_5_1_(1)_ESPCA-09457f_glider_gun.rle
  Gives a 4-way glider absorber and gun in ESPCA-09457f.
- Ch_5_5_1_(2)_ESPCA-098aef_glider_gun.rle
  Gives a 4-way glider absorber and gun in ESPCA-098aef.
- Ch_5_5_2_(1)_ETPCA-0347_3w_glider_gun.rle
  Gives a 3-way glider absorber and gun in ETPCA-0347.
- Ch_5_5_2_(2)_ETPCA-0347_6w_glider_gun.rle
  Gives a 6-way glider gun that appears from a disordered pattern in ETPCA-0347.
- Ch_5_6_1_(1)_ESPCA-0f00f0_Linear_Irrev_Self-repl.rle
  Shows self-replication of a pattern in the irreversible linear ESPCA-0f00f0.
- Ch_5_6_1_(2)_ESPCA-05f050_Linear_Irrev_Self-repl.rle
  Shows self-replication of a pattern in the irreversible linear ESPCA-05f050.
- Ch_5_6_1_(3)_ESPCA-095f60_Linear_Irrev_Self-repl.rle
  Shows self-replication of a pattern in the irreversible linear ESPCA-095f60.
- Ch_5_6_1_(4)_ESPCA-0af0a0_Linear_Irrev_Self-repl.rle
  Shows self-replication of a pattern in the irreversible linear ESPCA-0af0a0.
- Ch_5_6_1_(5)_ESPCA-03afc0_Linear_Irrev_Carpet.rle
  Generates carpet-like patterns in the irreversible linear ESPCA-03afc0.
- Ch_5_6_1_(6)_ESPCA-0d3a8f_Linear_Rev_Fractal.rle
  Generates fractal-like patterns in the reversible linear ESPCA-0d3a8f.
- Ch_5_6_1_(7)_ESPCA-07ca2f_Linear_Rev_Fractal.rle
  Generates fractal-like patterns in the reversible linear ESPCA-07ca2f.
- Ch_5_6_2_(1)_ESPCA-0dca8f_Fractal.rle
  Generates fractal-like patterns in the reversible ESPCA-0dca8f.
- Ch_5_6_2_(2)_ESPCA-0925bf_Disk.rle
  Generates disk-like patterns in the reversible ESPCA-0925bf.
- Ch_5_6_2_(3)_ESPCA-01eacf_Disk.rle
  Generates disk-like patterns in the reversible ESPCA-01eacf.
- Ch_5_6_3 (1)_ETPCA-0707_Linear_Irrev_Self-repl.rle
  Shows self-replication of a pattern in the irreversible linear ETPCA-0707.
- Ch_5_6_3_(2)_ETPCA-0330_Linear_Irrev_Snowflake.rle
  Generates snowflake-like patterns in the irreversible linear ETPCA-0330.

- Ch_5_6_3_(3)_ETPCA-0330_Linear_Irrev_Self-repl.rle
  Shows self-replication of a pattern in the irreversible linear ETPCA-0330.
- Ch_5_6_3_(4)_ETPCA-0550_Linear_Irrev_Fractal.rle
  Generates fractal-like patterns in the irreversible linear ETPCA-0550.
- Cx_Exercise_5_4_ETPCA-0347_12w_glider_gun.rle
  Gives a 12-way glider gun in ETPCA-0347.

Chapter 6. Making Reversible Turing Machines in Reversible ESPCAs

- Ch_6_1_(1)_ESPCA-01c5ef_marker_shifting.rle
  Shows that a position marker is shifted by a glider-12 in ESPCA-01c5ef.
- Ch_6_1_(2)_ESPCA-01c5ef_RLEMs_2-3_2-4_3-10.rle
  Shows implementation methods of RLEMs 2-3, 2-4 and 3-10 in ESPCA-01c5ef.
- Ch_6_1_(3)_ESPCA-01c5ef_RE.rle
  Shows an implementation method of an RE in ESPCA-01c5ef.
- Ch_6_1_(4)_ESPCA-01c5ef_RTM_parity_by_RE.mc
  Shows a composing method of the RTM T_{parity} out of REs in ESPCA-01c5ef.
- Ch_6_2_(1)_ESPCA-01caef_RLEM_2-2.rle
  Shows an implementation method of RLEM 2-2 in ESPCA-01caef.
- Ch_6_2_(2)_ESPCA-01caef_RE.rle
  Shows an implementation method of an RE in ESPCA-01caef.
- Ch_6_2_(3)_ESPCA-01caef_RTM_square_by_RE.mc
  Shows a composing method of the RTM T_{square} out of REs in ESPCA-01caef.
- Ch_6_3_(1)_ESPCA-02c5df_I-gate_and_inv_I-gate.rle
  Shows implementation methods of an interaction gate (I-gate) and its inverse (I^{-1}-gate) in ESPCA-02c5df.
- Ch_6_3_(2)_ESPCA-02c5df_RE.rle
  Shows an implementation method of an RE using I-gates and I^{-1}-gates in ESPCA-02c5df.
- Ch_6_3_(3)_ESPCA-02c5df_RTM_prime_by_RE.mc
  Shows a composing method of the RTM T_{prime} out of REs in ESPCA-02c5df.
- Ch_6_4_(1)_ESPCA-02c5bf_RE.rle
  Shows an implementation method of an RE using I-gates and I^{-1}-gates in ESPCA-02c5bf.
- Ch_6_4_(2)_ESPCA-02c5bf_RTM_power_by_RE.mc
  Shows a composing method of the RTM T_{power} out of REs in ESPCA-02c5bf.
- Ch_6_4_(3)_ESPCA-02c5bf_Metacell_of_01caef.mc
  Shows a composing method of *metacells* that simulate ESPCA-01caef out of REs in ESPCA-02c5bf.
- Cx_Exercise_6_6_ESPCA-01caef_RTM_square_by_RE.mc
  Shows a pattern of T_{square} that has a different input value from that of Ch_6_2_(3)_ESPCA-01caef_RTM_square_by_RE.mc.
- Cx_Exercise_6_7_ESPCA-01c5ef_RLEM_2-2.rle
  Shows an implementation method of RLEM 2-2 in ESPCA-01c5ef.
- Cx_Exercise_6_8_ESPCA-01caef_RLEM_2-3.rle
  Shows an implementation method of RLEM 2-3 in ESPCA-01caef.
- Cx_Exercise_6_9_ESPCA-02c5df_RLEM_2-2_2-3.rle
  Shows implementation methods of RLEMs 2-2 and 2-3 in ESPCA-02c5df.

Chapter 7. Making Reversible Turing Machines in Reversible ETPCAs

- Ch_7_1_(1)_ETPCA-0347_marker_shifting.rle
  Shows that a position marker is shifted by a glider-6 in ETPCA-0347.
- Ch_7_1_(2)_ETPCA-0347_RLEM_4-31.rle
  Shows an implementation method of RLEM 4-31 in ETPCA-0347.
- Ch_7_1_(3)_ETPCA-0347_RLEM_2-17.rle
  Shows an implementation method of RLEM 2-17 in ETPCA-0347.
- Ch_7_1_(4)_ETPCA-0347_RTM_parity_by_4-31.mc
  Shows a composing method of the RTM T_{parity} out of RLEM 4-31 in ETPCA-0347.
- Ch_7_1_(5)_ETPCA-0347_RTM_power_by_4-31.mc
  Shows a composing method of the RTM T_{power} out of RLEM 4-31 in ETPCA-0347.
- Ch_7_1_(6)_ETPCA-0347_RTM_parity_by_2-17.mc
  Shows a composing method of the RTM T_{parity} out of RLEM 2-17 in ETPCA-0347.
- Ch_7_1_(7)_ETPCA-0347_glider_gun_and_absorber.rle
  Shows a composing method of glider gun and absorber in ETPCA-0347.
- Ch_7_2_(1)_ETPCA-034z_RLEM_2-2.rle
  Shows an implementation method of RLEM 2-2 in the partial ETPCA-034z.
- Ch_7_3_(1)_ETPCA-0157_flip_flop.rle
  Shows an implementation method of a flip-flop module in ETPCA-0157.
- Ch_7_3_(2)_ETPCA-0157_RLEM_4-31.rle
  Shows an implementation method of RLEM 4-31 in ETPCA-0157.
- Ch_7_3_(3)_ETPCA-0157_RTM_parity_by_4-31.mc
  Shows a composing method of the RTM T_{parity} out of RLEM 4-31 i in ETPCA-0157.
- Ch_7_4_(1)_ETPCA-013z_flip_flop.rle
  Shows an implementation method of a flip-flop module in the partial ETPCA-013z.
- Ch_7_4_(2)_ETPCA-013z_RLEM_4_31.rle
  Shows an implementation method of LREM 4-31 in the partial ETPCA-013z.
- Ch_7_4_(3)_ETPCA-013z_RTM_parity_by_4-31.mc
  Shows a composing method of the RTM T_{parity} out of RLEM 4-31 in the partial ETPCA-013z.
- Cx_Exercise_7_5_ETPCA-0347_RLEM_2-2.rle
  Shows an implementation method of RLEM 2-2 in ETPCA-0347.

Chapter 8. Making Reversible Counter Machines in a Reversible SPCA

- Ch_8_(1)_SPCA-81_P3_basic_elements.rle
  Shows five kinds of basic elements for composing RCMs in the 81-state reversible SPCA P_3.
- Ch_8_(2)_SPCA-81_P3_RCM_twice.rle
  Shows a composing method of the RCM M_{twice} in the SPCA P_3.
- Ch_8_(3)_SPCA-81_P3_RCM_exp.rle
  Shows a composing method of the RCM M_{exp} and its inverse in the SPCA P_3.
- Cx_Exercise_8_5_P3_B-turn.rle
  Shows an implementation method of a backward-turn module by other modules in the SPCA P_3.
- Cx_Exercise_8_6_P3_Slow_space_moving.rle
  Shows an implementation method of slowly space-moving patterns in the SPCA P_3.
- Cx_Exercise_8_7_P3_RCM_half.rle
  Shows a composing method of the RCM M_{half} in the SPCA P_3.

Chapter 9. Open Problems and Future Research Problems

- Cx_Exercise_9_1_P3_RCM_loop.rle
  Shows a composing method of a periodic pattern in the 81-state reversible SPCA P_3 such that the ratio (maximum diameter)/(minimum diameter) is very large.

3 Frameworks of reversible CAs and other reversible systems

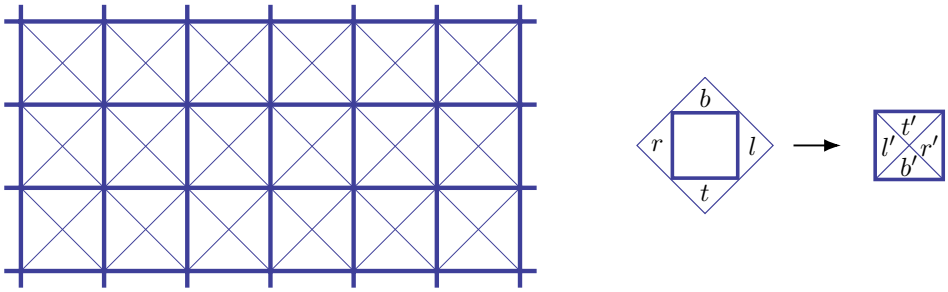
In this section, we briefly explain the frameworks of the CAs and other reversible computing models that are dealt with in the zip file. For more detailed explanations on these models, see the book *Reversible World of Cellular Automata* or [6].

3.1 Partitioned cellular automata (PCAs)

A *partitioned cellular automaton* (PCA) was first proposed in [7] to design reversible CAs. Here, we use 4-neighbor square partitioned cellular automata (SPCAs), and 3-neighbor triangular partitioned cellular automata (TPCAs).

3.1.1 4-neighbor square partitioned cellular automata (SPCAs)

In a PCA, each cell is divided into several parts, whose number is equal to the number of neighbor cells. A 2-dimensional 4-neighbor SPCA consists of an infinite number of square cells having four parts. Thus, its cellular space is as shown in Fig. 1 (a). Each part of a cell has its own state set. Hence, the set of states of one cell is the Cartesian product of the sets of states of the four parts. In an SPCA, a cell changes its state depending on the top part of the south-neighbor cell, the right part of the west cell, the bottom part of the north cell, and the left part of the east cell as shown in Fig. 1 (b). A *local function* f of a 4-neighbor SPCA is given by a set of local transition rules of the form $f(t,r,b,l) = (t',r',b',l')$ (Fig. 1 (b)).



(a) (b)

Figure 1: (a) Cellular space of a 4-neighbour SPCA, and (b) a local transition rule that represents $f(t,r,b,l) = (t',r',b',l')$.

Applying the local function f to all the cells simultaneously, a *global function* F is obtained. By the global function, configurations (*i.e.*, a state of the whole cellular space) evolve. If the global function is injective, then the PCA is called a *reversible PCA*. It is known that, in PCAs, the global function is injective, if and only if the local function is injective [6, 7]. Therefore, reversible CAs can be easily obtained by using the framework of PCAs.

An *elementary SPCA* (ESPCA) is the simplest subclass of SPCAs such that each part has the state set $\{0,1\}$ and the local function is rotation symmetric. Hence, a cell has 16 states. Since it is rotation symmetric, its local function is described by only six local transition rules. Figure 2 shows an example of a local function of a particular ESPCA defined by six local transition rules. The ESPCA given by Fig. 2 has the 6-digit hexadecimal identification number 01caef, which is obtained by reading the right-hand sides of the local transition rules as binary numbers.

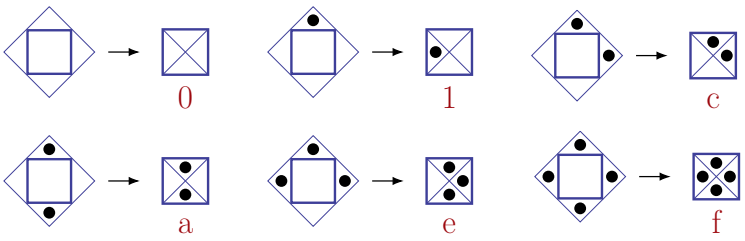


Figure 2: Six local transition rules that define the local function of ESPCA-01caef. Here, the state 0 and 1 of each part of a cell is represented by a blank and a particle (•).

We can see that there are 65,536 ESPCAs in total as shown in Fig. 3. An ESPCA with the identification number *uvwxyz* is denoted by ESPCA-*uvwxyz*. Its local and global functions are also denoted by f_{uvwxyz} and F_{uvwxyz}, respectively.

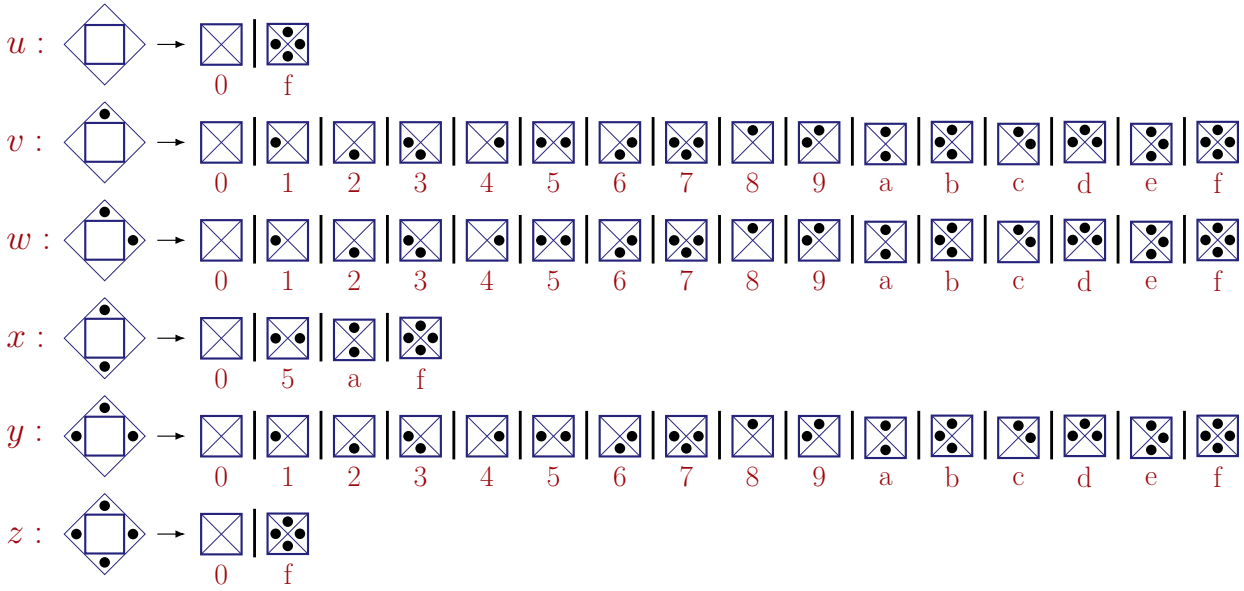


Figure 3: Expressing an ESPCA by a 6-digit hexadecimal identification number *uvwxyz*. Vertical bars indicate alternatives of the right-hand side of each local transition rule.

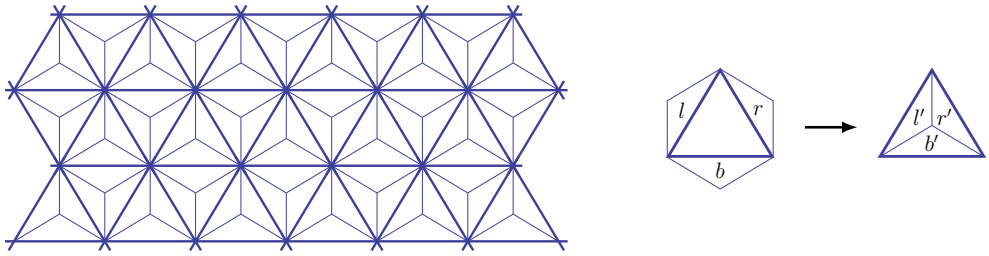
Remarks. When simulating an ESPCA in Golly, the states $0,\ldots,\mathrm{f}$ of a cell are represented by the states $0,\ldots,15$ in the Golly system. However, we use several additional states in Golly, which have the numbers starting from 16. These additional states are used to write "comments" in the cellular space. If complex objects, such as large logic elements or computing systems, are simulated in an ESPCA, it often becomes very difficult to recognize the states of the objects. In such a case, the additional states are used for indicating the macroscopic states of the objects. Of course, additional states must be placed so that they do not affect the operations of the original 16 states.

Also note that we use some different colors for the cell's states $0,\ldots,\mathrm{f}$ to obtain a visual effect. By this, particles also have different colors. However, of course, particles (state 1's) are the same state.

3.1.2 3-neighbor triangular partitioned cellular automata (TPCAs)

A 3-neighbor triangular partitioned cellular automaton (TPCA) is a PCA whose cell is triangular and is divided into three parts. The cellular space of a TPCA is shown in Fig 4 (a). Its local transition rule $f(l,b,r)=(l',b',r')$ is depicted by Fig. 4 (b). Though all the cells are identical in their logical operations, there are two kinds of directions, *i.e.*, upward and downward. Therefore, neighbor cells

of an up-triangle cell are different from those of a down-triangle cell. To up-triangle cells, the local transition rule shown in Fig. 4 (b) is applied. To down-triangle cells, the rule obtained by rotating both sides of Fig. 4 (b) by 180° is applied.



(a) (b)

Figure 4: (a) Cellular space of a 3-neighbour TPCA, and (b) a local transition rule that represents $f(l,b,r) = (l',b',r')$.

As in the case of an ESPCA, an *elementary TPCA* (ETPCA) is the simplest subclass of TPCAs such that each part has the state set $\{0,1\}$ and the local function is rotation symmetric. Hence, a cell has 8 states. Since it is rotation symmetric, its local function is described by only four local transition rules. Figure 5 shows an example of a local function of a particular ETPCA defined by four local transition rules. The ETPCA given by Fig. 5 has the 4-digit octal identification number 0347, which is obtained by reading the right-hand sides of the local transition rules as binary numbers.

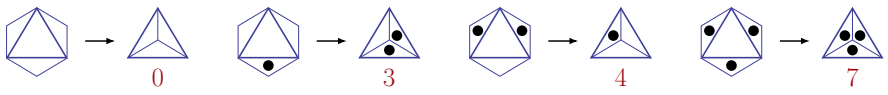


Figure 5: Four local transition rules that define the local function of ETPCA-0347.

There are 256 ETPCAs in total as shown in Fig. 6. An ETPCA with the identification number *wxyz* is denoted by ETPCA-*wxyz*. Its local and global functions are also denoted by f_{wxyz} and F_{wxyz}, respectively.

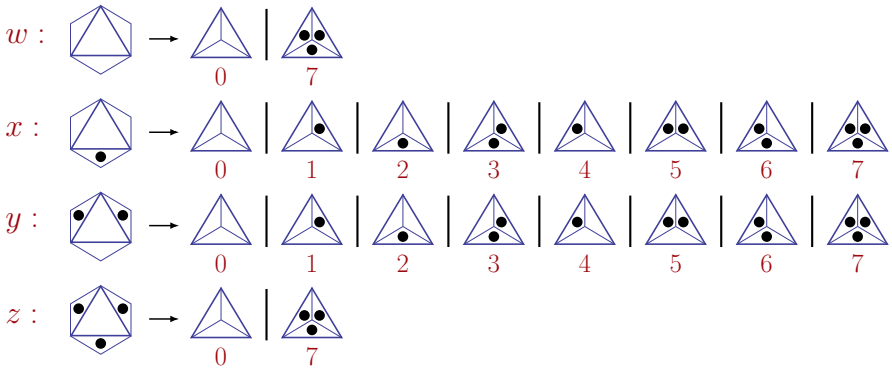


Figure 6: Expressing an ETPCA by a 4-digit octal identification number *wxyz*.

Remarks. In Golly, TPCAs are simulated on a square lattice. Figure 7 shows the x-y coordinates of each cell. Here, we assume the following: If $x+y$ is even (odd, respectively), then the cell at (x,y) is an up-triangle (down-triangle) cell.

A cell of an ETPCA has 8 states. However, we use 15 states for simulating it in Golly. The state 0 (*i.e.*, the blank state) of both up-triangle and down-triangle cells of an ETPCA is represented

by 0 in Golly. The states $1,\ldots,7$ of an up-triangle cell of an ETPCA are represented by $1,\ldots,7$ in Golly, respectively. On the other hand, the states $1,\ldots,7$ of a down-triangle cell of an ETPCA are represented by $8,\ldots,14$ in Golly, respectively. Therefore, if $x+y$ is even (odd, respectively), then the cell at (x,y) must have a state in $\{0,1,\ldots,7\}$ ($\{0,8,\ldots,14\}$) in Golly. Otherwise, it is not correctly simulated.

As in the case of SPCAs, we use some additional states starting from 15. They are for writing comments or indicating macroscopic states of large objects in the cellular space of Golly.

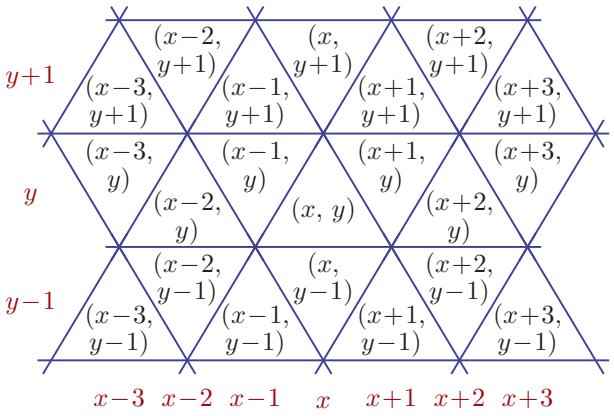


Figure 7: The x-y coordinates in the cellular space of TPCA. If $x+y$ is even, the cell at (x,y) is an up-triangle cell.

3.2 Reversible Turing machines (RTMs)

A one-tape Turing machine (TM) consists of a finite control, a read-write head, and a two-way infinite tape divided into squares in which symbols are written as shown in Fig. 8.

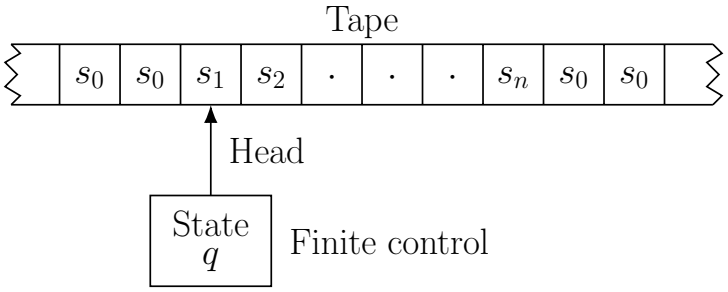


Figure 8: One-tape Turing machine (TM).

Movement of a TM is specified by a set of *quintuples* as in the following definition.

Definition 1 A *one-tape Turing machine* (TM) is defined by

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

where Q is a non-empty finite set of states, S is a non-empty finite set of tape symbols, q_0 is an *initial state* ($q_0 \in Q$), F is a set of *final states* ($F \subseteq Q$), and s_0 is a special *blank symbol* ($s_0 \in S$). Here, δ is a move relation, which is a subset of $(Q \times S \times S \times \{L,N,R\} \times Q)$. The symbols "$L$", "$N$", and "$R$" are *shift directions* of the head, which stand for "left-shift", "no-shift", and "right-shift", respectively. Each element of δ is a *quintuple* of the form $[p,s,s',d,q]$, which is called a *rule* of T. It means if T reads the symbol s in the state p, then write s', shift the head to the direction d, and go to the state q.

Determinism and reversibility of a TM is defined as below.

Definition 2 Let $T = (Q, S, q_0, F, s_0, \delta)$ be a TM. We call T a *deterministic TM*, if the following holds for any pair of distinct quintuples $[p_1, s_1, t_1, d_1, q_1]$ and $[p_2, s_2, t_2, d_2, q_2]$ in δ.

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

It means that for any pair of distinct rules, if the present states are the same, then the read symbols are different.

Here, we consider only deterministic TMs. Therefore, the term "deterministic" is omitted.

Definition 3 Let $T = (Q, S, q_0, F, s_0, \delta)$ be a TM. We call T a *reversible TM* (RTM), if the following holds for any pair of distinct quintuples $[p_1, s_1, t_1, d_1, q_1]$ and $[p_2, s_2, t_2, d_2, q_2]$ in δ.

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

It means that for any pair of distinct rules, if the next states are the same, then the shift directions are the same, and the written symbols are different.

It is known that any irreversible TM is simulated by a reversible TM that produce no garbage information when it halts [1]. Therefore, the class of RTMs is computationally universal.

Example 1 An RTM T_{parity} defined below is a very simple example.

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

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

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

It is easy to see that T_{parity} is reversible. Consider the pair of rules $[q_0,0,1,R,q_1]$ and $[q_2,1,0,R,q_1]$. The next states in these rules are the same (*i.e.*, q_1). We can see the shift directions in them are the same (*i.e.*, R), and the written symbols are different (*i.e.*, 1 and 0). Thus the pair satisfies the reversibility condition in Definition 3. No other pair of distinct rules have the same next state. Therefore T_{parity} is reversible. Complete computing processes starting from q_0011 and q_00111 are as follows.

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

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

Remarks. The file "Reversible_World_of_CAs.zip" contains a rule file and pattern files by which one-tape 2-symbol Turing machines can be simulated. Note that they can simulate any TM even if it is *irreversible*. Therefore, if one has designed a reversible 2-symbol TM, he/she must check its reversibility by referring Definition 3.

3.3 Reversible counter machines (RCMs)

A *counter machine* (CM) is defined as a kind of multi-tape TM as shown in Fig. 9. The tapes are read-only ones, and one-way infinite. The leftmost square of a tape contains the symbol Z, while all the other squares contain P. Therefore, if the machine reads the symbol Z (P, respectively), then it knows the contents of the counter is zero (positive). The increment and decrement operations on a counter are performed by shifting the corresponding head. There are two kinds of formulations of CMs. They are the quadruple form, and the program form.

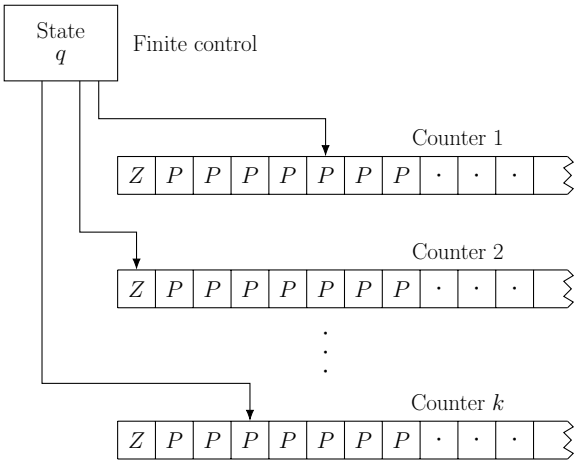


Figure 9: k-counter machine (CM(k)).

3.3.1 CM in the quadruple form

Definition 4 A *k-counter machine in the quadruple form* (CM(k)) is defined by

$$M = (Q, k, \delta, q_0, F).$$

Here, Q is a non-empty finite set of states. The integer k ($\in \{1,2,\ldots\}$) is the number of counters (*i.e.*, tapes). Thus M is also called a *k-counter machine* (CM(k)). The state q_0 is an *initial state* ($q_0 \in Q$), and F is a set of *final states* ($F \subseteq Q$). The CM M uses $\{Z,P\}$ as a tape alphabet. The symbol Z is written only on the leftmost square of each tape, while the symbol P is written in all other squares. The item δ is a move relation, which is a subset of $(Q \times \{1,\ldots,k\} \times \{Z,P\} \times Q) \cup (Q \times \{1,\ldots,k\} \times \{-1,0,+1\} \times Q)$. The symbols -1, 0 and $+1$ are decrement, no-change and increment operations on a counter, respectively. They are performed by left-shift, no-shift and right-shift operations of a tape head. In the following, -1 and $+1$ are also indicated by $-$ and $+$ for simplicity. Each element of δ is a *quadruple* or a *rule* of the form $[p,i,t,q] \in (Q \times \{1,\ldots,k\} \times \{Z,P\} \times Q)$, or $[p,i,d,q] \in (Q \times \{1,\ldots,k\} \times \{-,0,+\} \times Q)$. The quadruple $[p,i,t,q]$ is called a *counter-test rule*, and means that if M is in the state p, and the head of the i-th counter reads the symbol t, then go to the state q. The quadruple $[p,i,d,q]$ is called a *count-up/down rule*, and means that if M is in the state p, then shift the i-th head to the direction d by one square, and go to the state q. We assume each state $q_f \in F$ is a *halting state*, *i.e.*, there is no quadruple of the form $[q_f,i,x,q]$ in δ.

Definition 5 Let $M = (Q,k,\delta,q_0,F)$ be a CM. M is called a *deterministic CM*, if the following condition holds for any pair of distinct quadruples $[p_1,i_1,x_1,q_1]$ and $[p_2,i_2,x_2,q_2]$ in δ, where $D = \{-,0,+\}$.

$$(p_1 = p_2) \Rightarrow (i_1 = i_2 \wedge x_1 \notin D \wedge x_2 \notin D \wedge x_1 \neq x_2)$$

It means that for any pair of distinct rules, if the present states are the same, then they are both counter-test rules on the same counter, and the read symbols are different.

In the following, we deals with only deterministic CMs. Therefore the term "deterministic" is omitted. Reversibility of a CM is defined as follows.

Definition 6 Let $M = (Q, k, \delta, q_0, F)$ be a CM. We call M a *reversible CM* (RCM), if the following holds for any pair of distinct quadruples $[p_1, i_1, x_1, q_1]$ and $[p_2, i_2, x_2, q_2]$ in δ.

$$(q_1 = q_2) \Rightarrow (i_1 = i_2 \wedge x_1 \notin D \wedge x_2 \notin D \wedge x_1 \neq x_2)$$

It means that for any pair of distinct rules, if the next states are the same, they are both counter-test rules on the same counter, and the read symbols are different.

It is known that any TM can be simulated by a reversible CM with only two counters [3]. Therefore, the class of RCMs with two counters is computationally universal.

Example 2 We give a simple example: RCM(2) M_{twice}.

$$\begin{array}{rcl} M_{\text{twice}} & = & (\{q_0,q_1,q_2,q_3,q_4,q_5,q_6\}, 2, \delta_{\text{twice}}, q_0, \{q_6\}) \\ \delta_{\text{twice}} & = & \{ \quad [q_0,2,Z,q_1], \quad [q_1,1,Z,q_6], \quad [q_1,1,P,q_2], \quad [q_2,1,-,q_3], \\ & & \quad [q_3,2,+,q_4], \quad [q_4,2,+,q_5], \quad [q_5,2,P,q_1] \quad \} \end{array}$$

It is easy to verify that M_{twice} is deterministic and reversible. It computes the function $g(x) = 2x$. More precisely, $(q_0,(x,0)) \mid\!\!\overset{*}{-}_{M_{\text{twice}}} (q_6,(0,2x))$ holds for all $x \in \mathbb{N}$. For example, the complete computing process of M_{twice} for $x = 2$ is as follows.

$$\begin{array}{lllll} & (q_0,(2,0)) & \vdash_{M_{\text{twice}}} (q_1,(2,0)) & \vdash_{M_{\text{twice}}} (q_2,(2,0)) & \vdash_{M_{\text{twice}}} (q_3,(1,0)) \\ \vdash_{M_{\text{twice}}} (q_4,(1,1)) & \vdash_{M_{\text{twice}}} (q_5,(1,2)) & \vdash_{M_{\text{twice}}} (q_1,(1,2)) & \vdash_{M_{\text{twice}}} (q_2,(1,2)) & \vdash_{M_{\text{twice}}} (q_3,(0,2)) \\ \vdash_{M_{\text{twice}}} (q_4,(0,3)) & \vdash_{M_{\text{twice}}} (q_5,(0,4)) & \vdash_{M_{\text{twice}}} (q_1,(0,4)) & \vdash_{M_{\text{twice}}} (q_6,(0,4)) & \end{array}$$

3.3.2 CM in the program form

A CM in the program form was given in *Reversible World of Cellular Automata*. It was shown that it exactly characterizes the class of reversible CMs in the quadruple form.

Instructions for a CM(k) are the following: $I_i, D_i, B_i(b_0,b_1), M_i(m_0,m_1)$, and H, where $i \in \{1,\ldots,k\}$, and b_0, b_1, m_0 and m_1 are addresses of instructions. Meanings of the instructions are as follows.

I_i	Increment the i-th counter
D_i	Decrement the i-th counter
$B_i(b_0,b_1)$	Branch on the contents of the i-th counter, *i.e.*, if the i-th counter is 0, then go to b_0, else go to b_1
$M_i(m_0,m_1)$	Merge on the contents of the i-th counter, *i.e.*, if the i-th counter is 0, then merge from m_0, else from m_1
H	Halt

We give the sets $A^L, \mathbf{B}_k^L$ and $\mathbf{M}_k^L$ as follows, where L (> 0) is the length of a program.

$$\begin{array}{ll} A^L & = \{0,1,\ldots,L-1\} \\ \mathbf{B}_k^L & = \{B_i(b_0,b_1) \mid b_0,b_1 \in A^L \cup \{-\},\ i \in \{1,\ldots,k\}\} \\ \mathbf{M}_k^L & = \{M_i(m_0,m_1) \mid m_0,m_1 \in A^L \cup \{-\},\ i \in \{1,\ldots,k\}\} \end{array}$$

Here, A^L is the set of *addresses* of instructions, where the 0-th instruction has the address 0, and the last has $L-1$. The set $\mathbf{B}_k^L$ ($\mathbf{M}_k^L$, respectively) contains all possible $B_i(b_0,b_1)$ instructions ($M_i(m_0,m_1)$ instructions), where $-$ means no address is specified. If $b_p \in A^L$ ($m_p \in A^L$, respectively) for $p \in \{0,1\}$, it is called a *destination address* (*source address*) of *port* p of the instruction. The set $\mathbf{S}_k^L$ of instructions, which is for a program of length L of CM(k), is as follows.

$$\mathbf{S}_k^L = \{I_i, D_i \mid i \in \{1,\ldots,k\}\} \cup \mathbf{B}_k^L \cup \mathbf{M}_k^L \cup \{H\}$$

Definition 7 A *well-formed program* (WFP) P of length L for CM(k) is a mapping $P : A^L \to \mathbf{S}_k^L$ that satisfies the following syntactic constraints.

(C1) The last instruction must be H or B_i instruction:

$$P(L-1) \in \{H\} \cup \mathbf{B}_k^L$$

(C2) The 0-th instruction must not be an M_i instruction, and the instruction just before M_i must be an H or B_i instruction:

$$P(0) \notin \mathbf{M}_k^L \ \wedge \ \forall a \in A^L - \{0\}(P(a) \in \mathbf{M}_k^L \Rightarrow P(a-1) \in \{H\} \cup \mathbf{B}_k^L)$$

(C3) If the instruction of the address a is B_i, and its port p has a destination address a', then the instruction at the address a' must be M_i, and its port p has the source address a:

$$\begin{array}{l} \forall a, a' \in A^L, \ \forall p \in \{0,1\}, \ \forall i \in \{1,\ldots,k\}, \\ \forall b_0, b_1 \in A^L \cup \{-\}, \ \exists m_0, m_1 \in A^L \cup \{-\} \\ \quad ((P(a) = B_i(b_0,b_1) \wedge b_p = a') \Rightarrow (P(a') = M_i(m_0,m_1) \wedge m_p = a)) \end{array}$$

(C4) If the instruction of the address a is M_i, and its port p has a source address a', then the instruction at the address a' must be B_i, and its port p has the destination address a:

$$\begin{array}{l} \forall a, a' \in A^L, \ \forall p \in \{0,1\}, \ \forall i \in \{1,\ldots,k\}, \\ \forall m_0, m_1 \in A^L \cup \{-\}, \ \exists b_0, b_1 \in A^L \cup \{-\} \\ \quad ((P(a) = M_i(m_0,m_1) \wedge m_p = a') \Rightarrow (P(a') = B_i(b_0,b_1) \wedge b_p = a)) \end{array}$$

The constraint (C1) prevents the case of going to the address L. The constraint (C2) guarantees that each M_i instruction is activated only by B_i instructions. The constraints (C3) and (C4) say that the destination addresses of port p of B_i instructions, and the source addresses of port p of M_i instructions have one-to-one correspondence for each $p \in \{0,1\}$.

Example 3 Let P_{move} be the following sequence of instructions.

0	1	2	3	4	5	6	7	8
$B_2(1,-)$	$M_2(0,6)$	D_1	I_2	$B_1(7,5)$	$M_1(-,4)$	$B_2(-,1)$	$M_1(4,-)$	H

We can verify that P_{move} satisfies the constraints (C1) – (C4) in Definition 7. Therefore, it is a well-formed program (WFP) of a CM(2). Since it is cumbersome to follow addresses in B_i and M_i instructions, we often draw a WFP in a graphical form as in Fig 10.

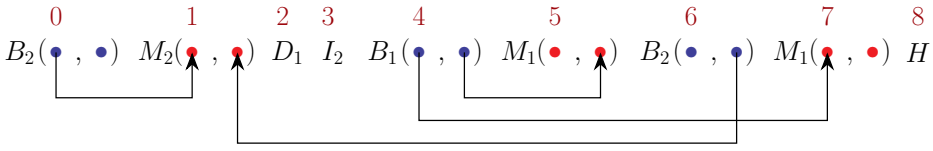


Figure 10: Graphical representation of of the WFP P_{move}.

We now define a CM M in the program form, which has a WFP.

Definition 8 A *CM in the program form* is defined by

$$M = (P, k, A_F),$$

where P is a WFP of length L, k is the number of counters, and A_F is a set of *final addresses* that satisfy the following: $A_F \subseteq \{a \mid a \in A^L \wedge P(a) = H\}$, where $A^L = \{0,\ldots,L-1\}$.

Remarks. The file "Reversible_World_of_CAs.zip" contains a rule file and pattern files by which CMs can be simulated. Note that they can simulate a CM in the quadruple form even if it is *irreversible*. Therefore, if one has designed a reversible CM, he/she must check its reversibility by Definition 6.

3.4 Reversible logic element with memory (RLEM)

A *reversible logic element with memory* (RLEM), which was first introduced in [4], is an important logic element for implementing universal reversible computing systems in a reversible CA. This is because RTMs and RCMs can be constructed out of RLEMs more easily than to use reversible logic gates. An RLEM is a kind of a *reversible finite automaton* having output symbols as well as input symbols, which is called a reversible sequential machine of Mealy type.

Definition 9 A *sequential machine* (SM) is defined by $M = (Q, \Sigma, \Gamma, \delta)$, where Q is a finite set of states, Σ and Γ are finite sets of input and output symbols, and $\delta : Q \times \Sigma \to Q \times \Gamma$ is a move function (Fig. 11 (a)). If δ is injective, it is called a *reversible sequential machine* (RSM).

Definition 10 A *reversible logic element with memory* (RLEM) is an RSM $M = (Q, \Sigma, \Gamma, \delta)$ that satisfies $|\Sigma| = |\Gamma|$. M is called a $|Q|$-state $|\Sigma|$-symbol RLEM.

To use an SM as a logic element, we interpret it as the one with decoded input ports and output ports (Fig. 11 (b)). Namely, for each input symbol, there is a unique input port to which a signal (or a particle) is given. It is also the case for the output symbols. Therefore, signals should not be given to two or more input ports at the same time.

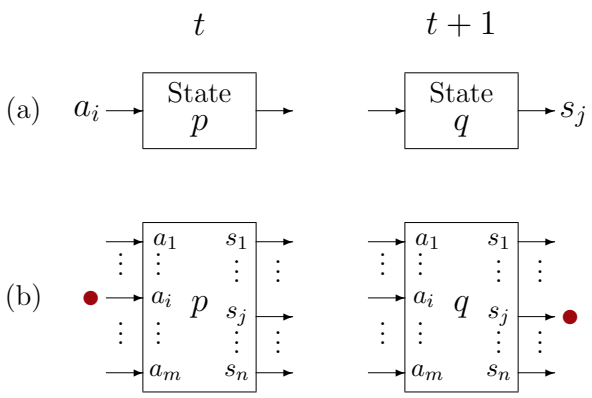


Figure 11: (a) A sequential machine (SM) with $\delta(p, a_i) = (q, s_j)$, and (b) an interpretation of it as a module having decoded input ports and output ports

3.4.1 2-state RLEMs

RLEMs having two states are particularly important. Despite its simplicity they are very powerful in composing reversible computers.

We first give an *identification number* to each of 2-state k-symbol RLEMs. Let $M = (Q, \Sigma, \Gamma, \delta)$ be an RLEM such that $|Q| = 2$ and $|\Sigma| = |\Gamma| = k$. Since δ is a bijection, there are $(2k)!$ kinds of δ's. Here, we fix the sets Q, Σ and Γ as follows: $Q = \{0, 1\}$, $\Sigma = \{a_1, \ldots, a_k\}$ and $\Gamma = \{s_1, \ldots, s_k\}$. Thus, a bijection δ can be specified by a permutation on the set $\{0, 1\} \times \{s_1, \ldots, s_k\}$. All the move functions δ's of 2-state k-symbol RLEMs are numbered by $0, \ldots, (2k)! - 1$ in the lexicographic order of permutations. An identification number is obtained by attaching the prefix "k-" to the serial number, *e.g.* 4-289. The RLEM with an identification number k-n is denoted by RLEM k-n.

Next, we show a method of representing a move function of a 2-state RLEM graphically. It makes it easy to describe a move function. Here, the method is explained using the following example (its detailed definition is given in [6]).

Example 4 Consider RLEM 4-289: $M_{4\text{-}289} = (\{0,1\}, \{a_1,a_2,a_3,a_4\}, \{s_1,s_2,s_3,s_4\}, \delta_{4\text{-}289})$. Its move function $\delta_{4\text{-}289}$ is described in Table 1. Figure 12 shows a graphical representation of $\delta_{4\text{-}289}$. In this figure, each of the two states are represented by a rectangle having input ports and output ports. The relation between the inputs and the outputs is indicated by solid and dotted lines. We assume a signal (or a particle) is given to at most one input port at a time. If a particle is given to an input port, it moves along the line connected to the input port, and goes out from the output port connected to it. If a particle goes through a dotted line, the state does not change (Fig. 13 (a)). On the other hand, if it goes through a solid line, the state changes to the other (Fig. 13 (b)).

Table 1: Move function $\delta_{4\text{-}289}$.

	Input			
Present state	a_1	a_2	a_3	a_4
0	0 s_1	0 s_2	1 s_1	1 s_2
1	0 s_3	0 s_4	1 s_4	1 s_3

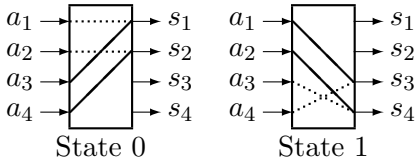


Figure 12: Graphical representation of RLEM 4-289.

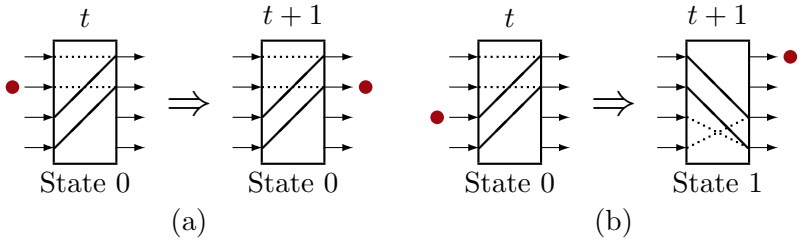


Figure 13: (a) If a particle passes a dotted line, the state remains to be the same. (b) If a particle passes a solid line, the state changes to the other.

There are $(2k)!$ kinds of 2-state k-symbol RLEMs. However, among them there are many equivalent RLEMs, which are obtained by renaming the states and/or the input/output symbols. They are called *equivalent* RLEMs. Hence, the total number of essentially different 2-state k-symbol RLEMs decreases significantly.

The total numbers of different 2-state 2-, 3- and 4-symbol RLEMs are 24, 720 and 40320, respectively. On the other hand, the numbers of equivalence classes of 2-, 3- and 4-symbol RLEMs are 8, 24 and 82, respectively [8, 6]. Figure 14 shows all representative RLEMs in the equivalence classes of 2- and 3-symbol RLEMs. Each representatives are chosen so that it has the smallest identification number in the class.

Among RLEMs, there are some *degenerate* ones that are further equivalent to connecting wires, or to a simpler RLEM with fewer symbols. A more precise definition of degeneracy is found in [6]. In Fig. 14, degenerate ones are indicated at the upper-right corner of a box. For example, RLEMs 3-1, 3-6 and 3-21 are degenerate ones. RLEM 3-1 is equivalent to three connecting wires, since no state-change can occur. RLEM 3-6 is equivalent to RLEM 2-2 and one connecting wire. RLEM 3-21 is equivalent to three connecting wires, since two states have exactly the same input-output relation. Table 2 shows the classification result. Among all RLEMs, non-degenerate ones are the main concern of the study.

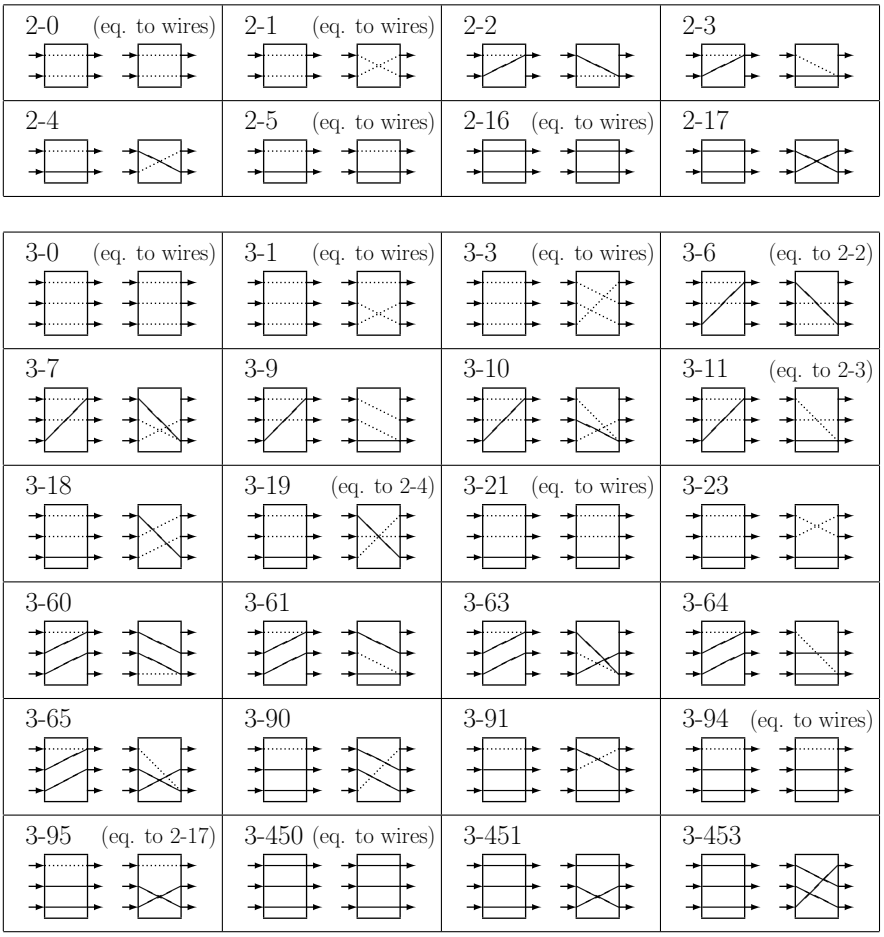


Figure 14: Representatives of equivalence classes of 2- and 3-symbol RLEMs [8, 6].

Table 2: Numbers of equivalence classes of degenerate and non-degenerate k-symbol RLEMs [6].

k	All equivalence classes	Degenerate ones	Non-degenerate ones
2	8	4	4
3	24	10	14
4	82	27	55

There are several useful RLEMs. The first one is a *rotary element* (RE), which was first proposed in [4]. It is equivalent to RLEM 4-289 (Fig. 12). However, we express it as in Fig 15. Intuitively, an RE has a rotatable bar inside, and an incoming signal is controlled by the bar. It takes either of the two states V or H, depending on the direction of the bar. If the direction of a coming signal is parallel to the bar, the signal goes straight ahead, and the state does not change (Fig. 16 (a)). If the direction of a coming signal is orthogonal to the bar, the signal turns right, and the state changes (Fig. 16 (b)). Hence, its operations are easily understood.

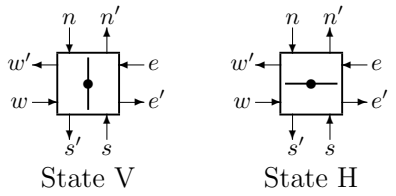


Figure 15: Two states V and H of a rotary element (RE).

It has been shown that any RTM can be constructed rather simply using only REs [4]. Figure 17 is an implementation of the RTM T_{parity} (Example 1) by REs.

The second one is RLEM 4-31 (Fig. 18). It is also possible to construct any RTM by it (Fig. 19).

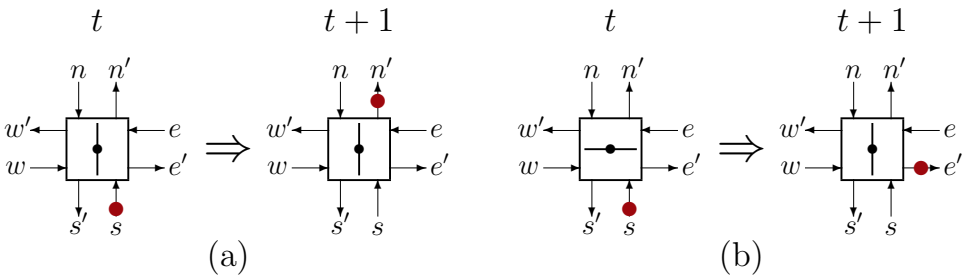


Figure 16: Operations of an RE. (a) Parallel case, and (b) orthogonal case.

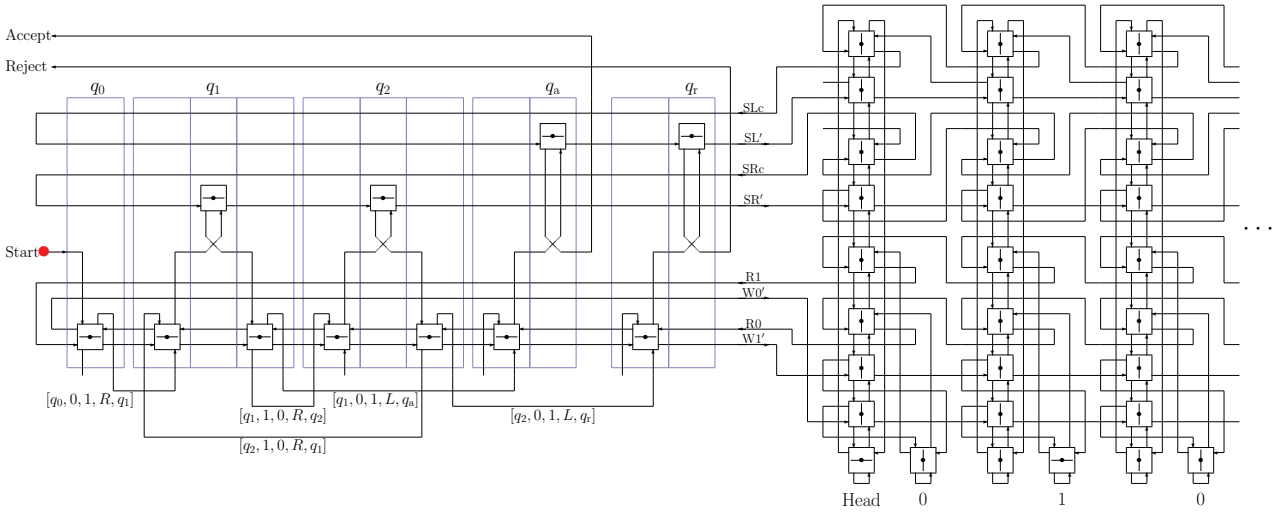


Figure 17: RTM T_{parity} composed of REs [6].

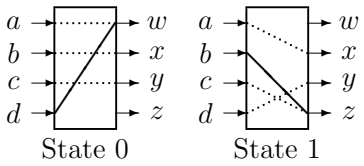


Figure 18: RLEM 4-31.

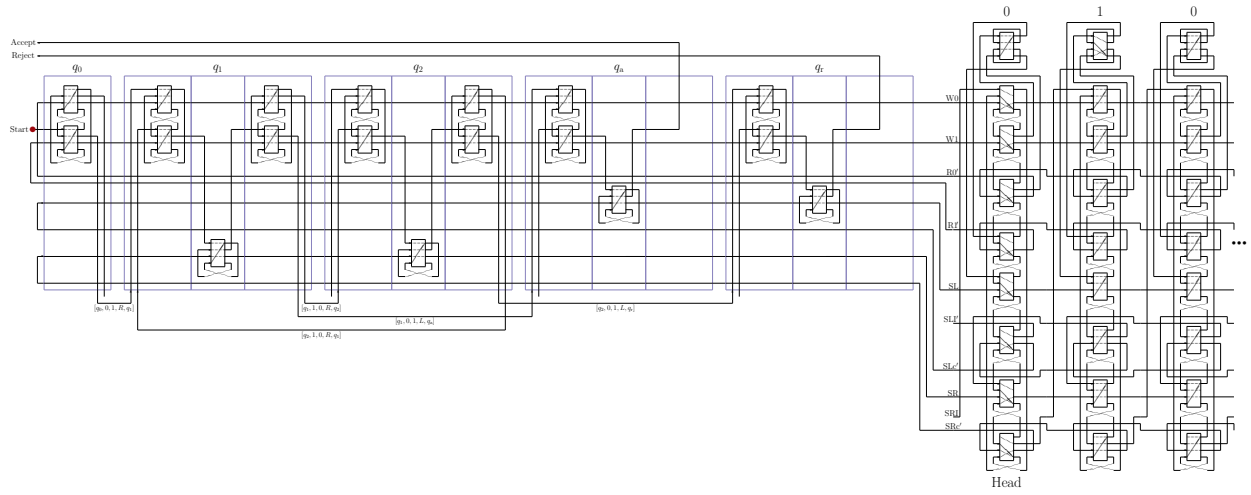


Figure 19: RTM T_{parity} composed of RLEM 4-31 [6].

Definition 11 An RLEM R is called *intrinsically universal* if any RSM M can be realized by a circuit composed only of R.

We can see that all non-degenerate 2-state RLEMs except only three are intrinsically universal. The result has been shown in the following way.

Proposition 1 [5] *An RE is intrinsically universal.*

Lemma 1 [2, 8] *RE is realizable by RLEM 3-10.*

Lemma 2 [2] *RLEM 3-10 is realizable by RLEMs 2-3 and 2-4.*

Lemma 3 [8] *RLEMs 2-3 and 2-4 are realizable by any one of 14 non-degenerate 2-state 3-symbol RLEMs.*

Lemma 4 [8] *Let M be an arbitrary non-degenerate 2-state k-symbol RLEM ($k \geq 3$). Then, there exists a non-degenerate 2-state $(k-1)$-symbol RLEM M' that is realizable by M.*

From Proposition 1 and Lemmas 1–4, we have the following proposition.

Proposition 2 [8] *All non-degenerate 2-state k-symbol RLEMs are intrinsically universal if $k \geq 3$.*

Furthermore, Matthew Cook and Ethan Palmiere proved the following.

Proposition 3 [Cook and Palmiere][1] *RLEM 2-17 is intrinsically universal.*

Figure 20 summarizes the above results. Thus, RTMs can be constructed out of any one of intrinsically universal RLEMs.

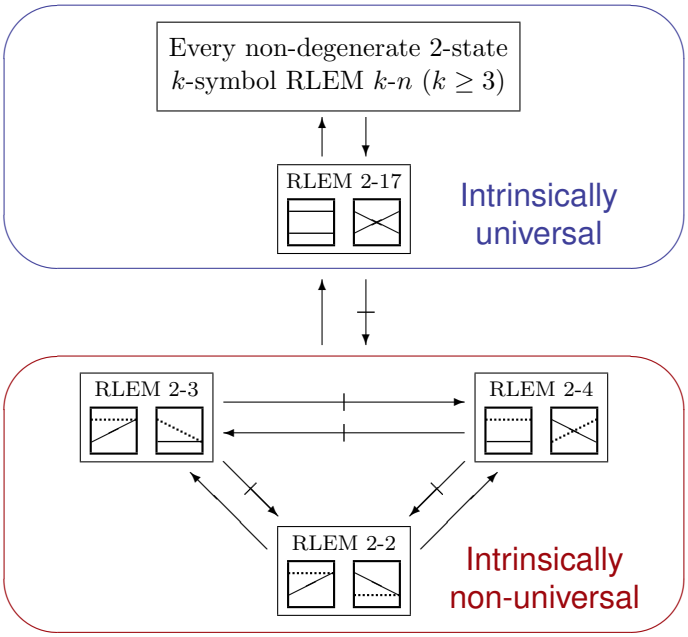


Figure 20: Intrinsic universality/non-universality of non-degenerate 2-state RLEMs. Here, $A \to B$ ($A \not\to B$, respectively) indicates that A is (is not) realizable by B. Note that the relations among the four 2-symbol RLEMs have been shown in [9].

[1] Personal communication.

Acknowledgements: I express my gratitude to the developing and support teams of *Golly* [10]. Without Golly, this work has not been accomplished.

References

[1] Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**, 525–532 (1973). doi:10.1147/rd.176.0525

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

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

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

[5] Morita, K.: A new universal logic element for reversible computing. In: Grammars and Automata for String Processing (eds. C. Martin-Vide, and V. Mitrana), pp. 285–294. Taylor & Francis, London (2003). doi:10.1201/9780203009642

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

[7] Morita, K., Harao, M.: Computation universality of one-dimensional reversible (injective) cellular automata. Trans. IEICE **E72**, 758–762 (1989). https://ir.lib.hiroshima-u.ac.jp/00048449

[8] Morita, K., Ogiro, T., Alhazov, A., Tanizawa, T.: Non-degenerate 2-state reversible logic elements with three or more symbols are all universal. J. Multiple-Valued Logic and Soft Computing **18**, 37–54 (2012)

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

[10] Trevorrow, A., Rokicki, T., Hutton *et al.*, T.: Golly: an open source, cross-platform application for exploring Conway's Game of Life and other cellular automata. https://golly.sourceforge.io/ (2005)