

Simulating a reversible elementary square partitioned cellular automaton with the ID number 01caef on Golly (Ver. 2)

Kenichi Morita* 

December 2021, revised in March 2022

Abstract

The rule files and the pattern files given here are the ones for emulating a reversible and conservative *elementary square partitioned cellular automaton* (ESPCA) with the hexadecimal ID number “01caef” on *Golly*, a general purpose CA simulator. Despite its simplicity of the local transition function, the ESPCA shows fascinating behavior. In particular, there exists a useful space-moving pattern called a *glider* in it. Colliding a glider with another pattern called a *blinker*, interesting phenomena appear. We observe that, using only these two patterns and three kinds of phenomena as basic operations, any reversible Turing machines (RTMs) can be constructed. Ten examples of patterns for *Golly*, a general purpose CA simulator, are given. In particular, whole computing processes of RTMs simulated in the ESPCA can be seen on it.

1 Elementary square partitioned cellular automaton (ESPCA)

A 4-neighbor *square partitioned cellular automaton* (SPCA) is a two-dimensional CA whose square cell is divided into four parts as in Fig. 1 (a). The next state of a cell is determined depending on the present states of the four adjacent parts of the neighboring cells as shown in Fig. 1 (b).

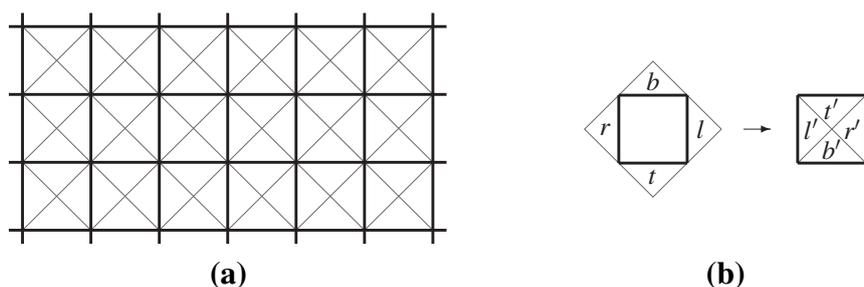


Figure 1: (a) Cellular space of a 4-neighbor square partitioned cellular automaton (SPCA), and (b) its local transition rule

An *elementary SPCA* (ESPCA) is a subclass of SPCAs such that its local transition function is rotation-symmetric, and each of four parts of a cell has only two states. Since an ESPCA is rotation-symmetric, its local transition function is defined by only six local transition rules, and hence very simple. Figure 2 shows a set of local transition rules of a particular ESPCA 01caef. The hexadecimal ID number 01caef is obtained by reading the dot patterns of the right-hand sides of the local rules as binary numbers. Since we consider this ESPCA here, we denote it by P_0 for short in the following.

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

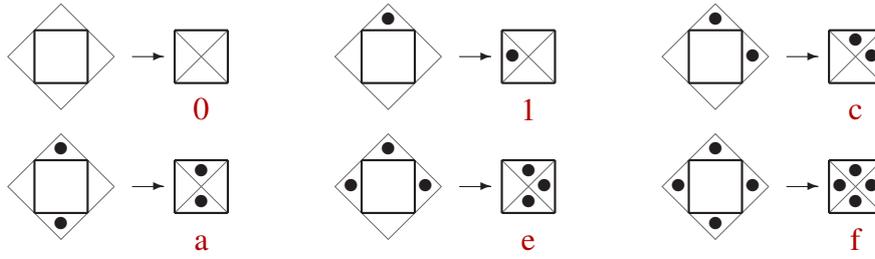


Figure 2: Local transition function defined by the six local transition rules of a particular reversible and conservative ESPCA 01caef, which is denoted by P_0 hereafter

The ESPAC P_0 is *reversible* since there is no pair of local transition rules that have the same right-hand sides as seen in Fig. 2. Hence, every configuration has exactly one predecessor configuration (see, e.g., [3] for the details of reversible CAs). The ESPCA P_0 is also *conservative*, since the number of particles is conserved in each local transition rule.

1.1 Useful patterns in ESPCA P_0

The pattern shown in Fig. 3 is called a *glider*. It moves diagonally by one cell in 12 steps.

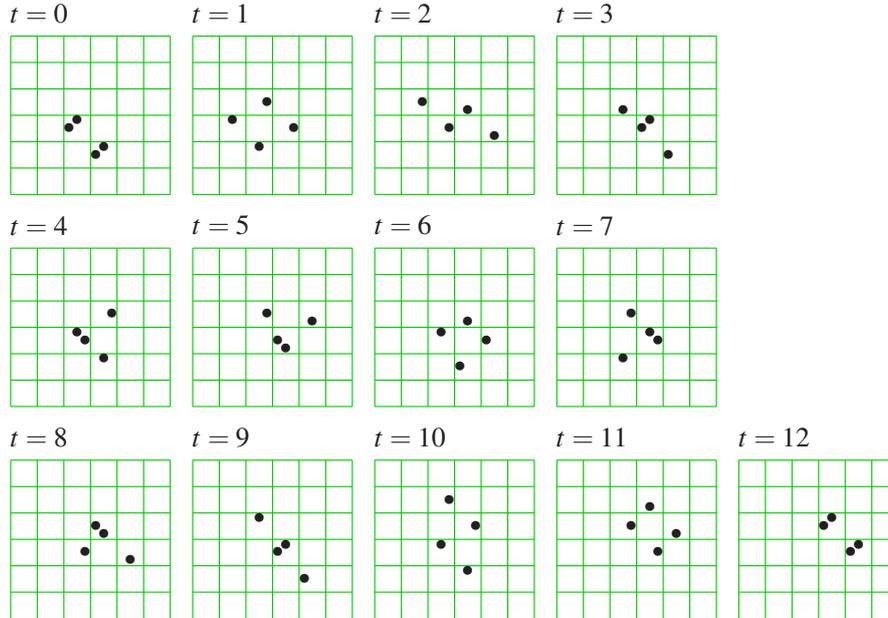


Figure 3: Glider in P_0

The pattern in Fig. 4 (a) is called a *blinker*. It is a periodic pattern of period 2. The pattern in Fig. 4 (b) is called a *block*. It is a *stable pattern*, i.e., a periodic pattern of period 1. A block will be used only for writing comments and indicating a border of a logic element in the cellular space.

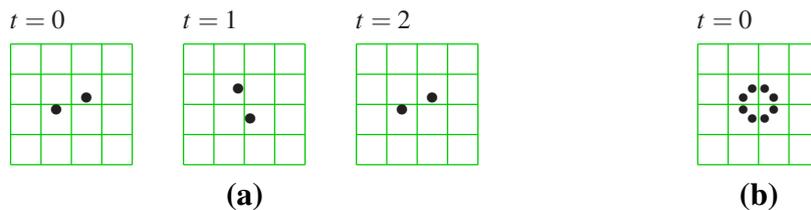


Figure 4: Periodic patterns in P_0 . (a) Blinker, and (b) block

1.2 Three useful phenomena in ESPCA P_0

Interacting a glider with a blinker, we can observe interesting phenomena. There are three useful phenomena. They are modularized as “gadgets”.

The first one is shown in Fig. 5. Colliding a glider with a blinker in this manner, a right-turn of a glider is realized.

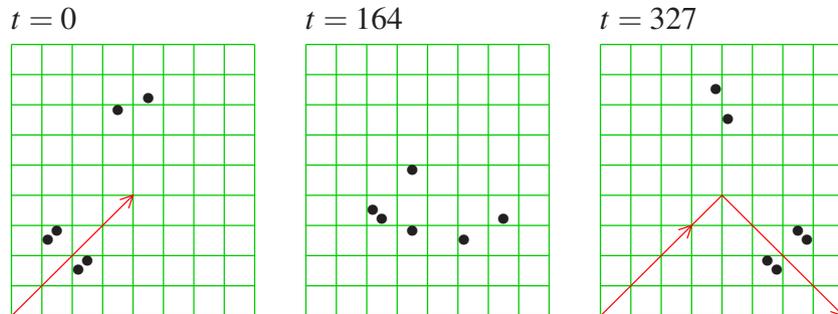


Figure 5: Right-turn gadget of a glider in P_0

The second is in Fig. 6. By this, a glider makes a U-turn. This operation is used to test if a blinker exists at a specified position. It is also used to reversibly merge two signal paths.

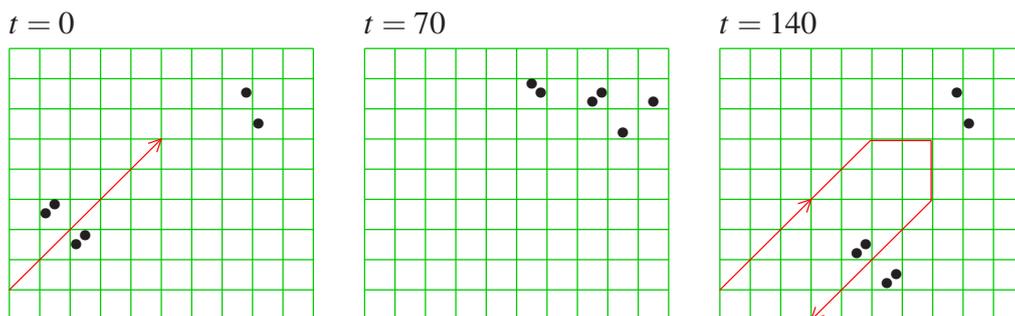


Figure 6: U-turn gadget of a glider in P_0

The third is in Fig. 7. By this, the position of the blinker is shifted by 6 cells, and the glider makes a right-turn. Using this operation, a kind of memory device is realized, where the memory states are kept by the positions of the blinker. It is also used to test if a blinker exists at a specified position.

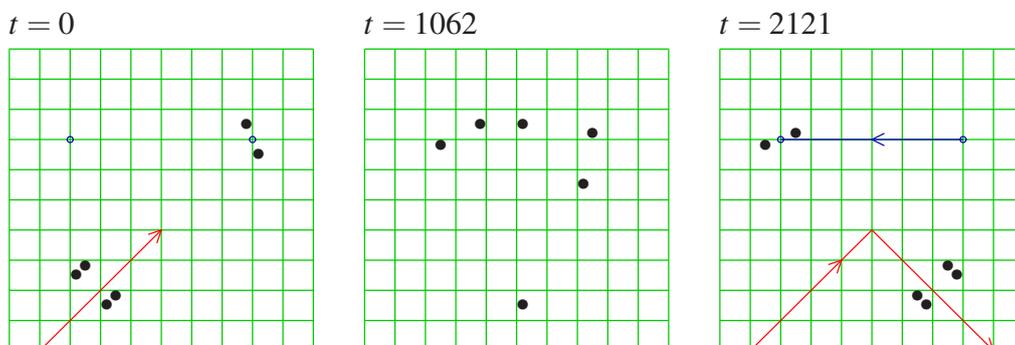


Figure 7: Shifting gadget of a blinker by a glider in P_0

2 Reversible logic element with memory (RLEM)

Using only three phenomena shown above, we can compose reversible Turing machines (RTMs). Here we employ a reversible logic element with memory (RLEM) [3] rather than a reversible logic gate as a logical primitive. By this, construction of RTMs is greatly simplified.

A *reversible logic element with memory* (RLEM) is a kind of a finite automaton having output symbols as well as input symbols. There are infinitely many 2-state RLEMs if we do not restrict the number of input/output symbols. It is known that every 2-state RLEM that has three or more I/O symbols is *universal*, which means any reversible finite automaton is composed only of it [3, 6]. In the following, we show two methods of using 2-state 4-symbol RLEMs for composing RTMs. One is based on a rotary element, a typical RLEM, and the other is to use RLEM 4-31.

2.1 Rotary element (RE), a typical RLEM

A *rotary element* (RE), first introduced in [2], is an RLEM shown in Fig. 8. Conceptually, it has a rotatable bar that controls the move direction of an incoming signal (or a particle). It takes one of the two states V and H depending on the direction of the bar (i.e., vertical or horizontal). If a signal comes from the direction parallel to the bar, it goes straight ahead and the state does not change. If a coming signal is orthogonal to the bar, it turns rightward and the state changes. It is reversible in the following sense: From the state at $t + 1$ and the output, the state at t and the input are uniquely determined. It has been shown that any reversible finite automaton can be constructed by using REs [3]. In this sense, it is a *universal* RLEM. It is also known that any reversible Turing machine can be constructed out of REs [2, 3].

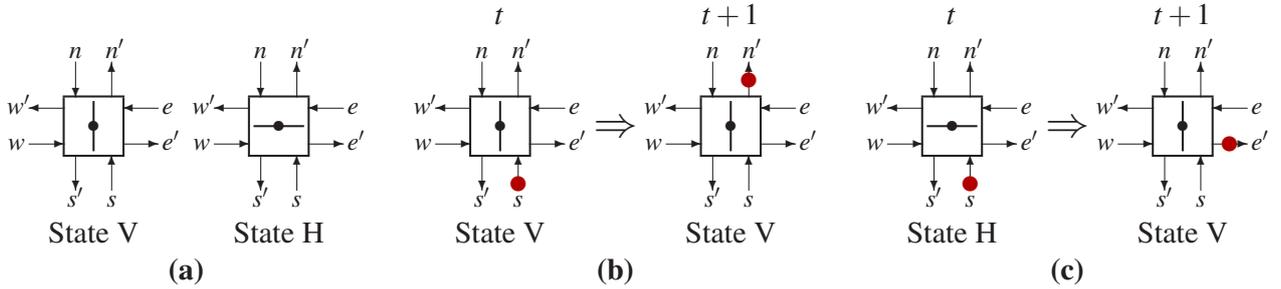


Figure 8: Rotary element (RE) and its operations. (a) Two states of RE. (b) The operation in the parallel case, and (c) that in the orthogonal case

RLEMs other than RE have different graphical representations. Fig. 9 shows that of RLEM 3-10, where “3” means that it has three I/O symbols, and “10” is its serial number in the class of 3-symbol RLEMs. Two boxes in Fig. 9 (a) indicate its two states. The dotted and solid lines give input-output relation in each state. If an input signal goes through a dotted line, the state does not change (Fig. 9 (b)). If it goes through a solid line, the state changes (Fig. 9 (c)). Note that RE can also be represented by such a figure, but we employ Fig. 8 for ease in understanding.

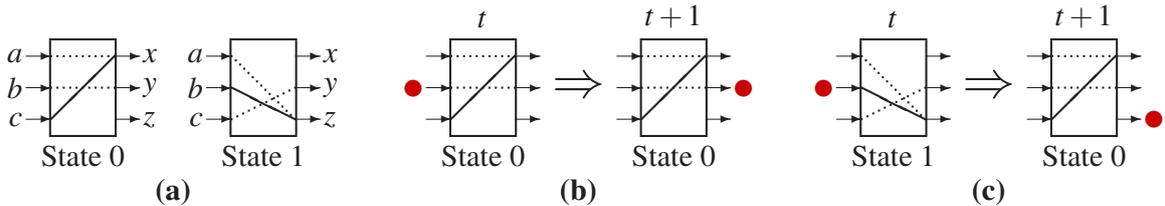


Figure 9: RLEM 3-10 and its operations. (a) Two states of RLEM 3-10. (b) The case where the state does not change, and (c) the case where the state changes

It is known that RE can be composed of RLEM 3-10 as shown in Fig. 10 [6]. Moreover, RLEM 3-10 is composed of RLEMs 2-3 and 2-4 as in Fig. 11 [1]. Hence, the set {RLEM 2-3, RLEM 2-4} is universal, though each of RLEM 2-3 and RLEM 2-4 is non-universal [8].

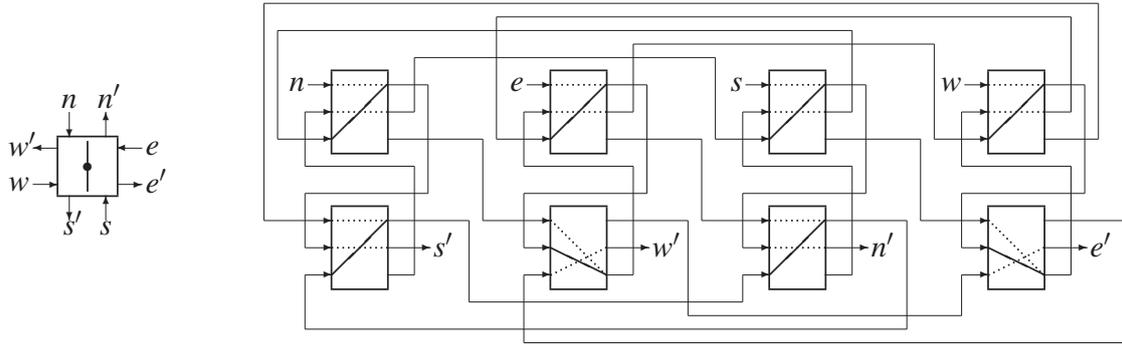


Figure 10: RE is simulated by a circuit composed of RLEM 3-10 [6]. This figure shows the state V of an RE. By complementing the states of the bottom four RLEMs, we have the state H of an RE

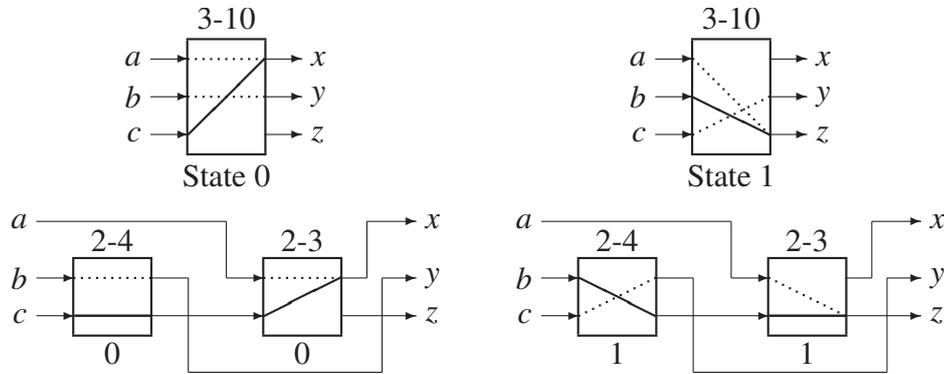


Figure 11: RLEM 3-10 is simulated by a circuit composed of RLEMs 2-3 and 2-4 [1]

In Sect.2.3.1, patterns of ESPCA P_0 that simulate RLEMs 2-3 and 2-4 are given. Therefore, RE is also realized in P_0 .

2.2 RLEM 4-31, another useful RLEM

Though the operations of RE is very easily understood, a pattern in P_0 that simulates RE becomes rather large. Therefore, we also use RLEM 4-31 (Fig. 12), which is useful for composing RTMs as shown in Fig. 23.

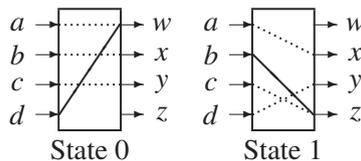


Figure 12: RLEM 4-31

2.3 Implementing RLEMs in ESPCA P_0

2.3.1 RLEMs 2-2, 2-3 and 2-4

We first consider how RLEMs 2-2, 2-3 and 2-4 (Fig. 13) are implemented in P_0 .

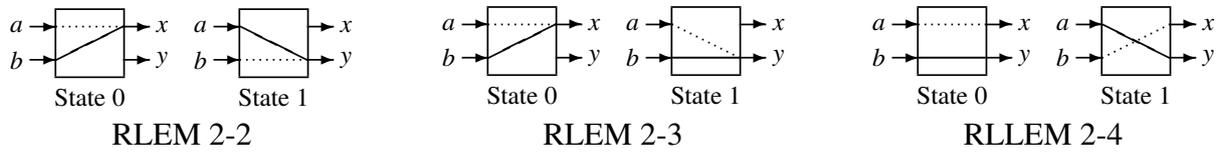


Figure 13: Three kinds of 2-symbol 2-state RLEMs.

RLEM 2-2

Fig. 14 shows a pattern that simulates RLEM 2-2 in P_0 . Although RLEM 2-2 is not universal [8], it is implemented rather simply. Therefore, it is convenient for explaining a method of designing a pattern that simulates an RLEM in P_0 firstly.

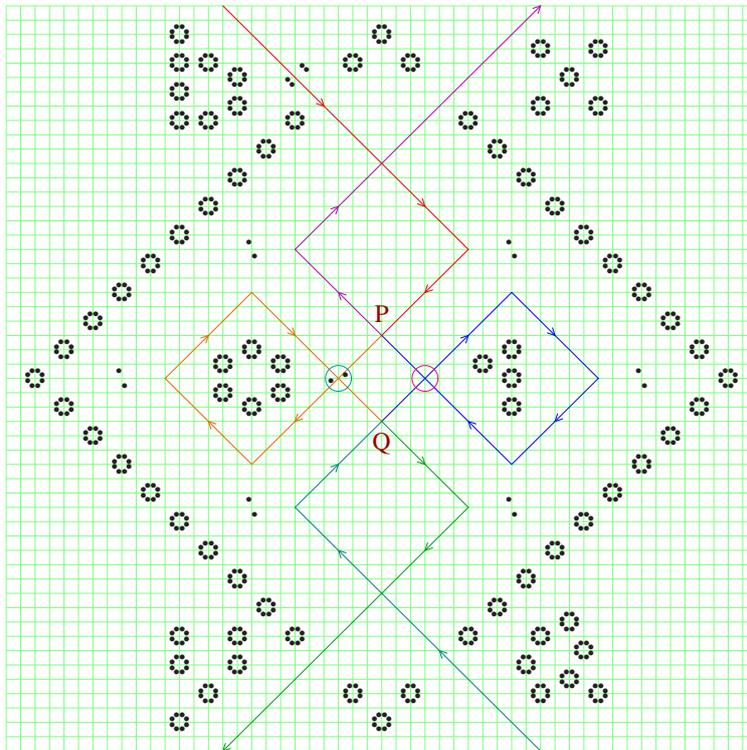


Figure 14: RLEM 2-2 implemented in P_0

There are seven blinkers in this pattern. One is used as a *position marker* for keeping the memory state 0 or 1, and six are used for turning a signal. Small two circles near the center of the pattern show possible positions of the marker. If the marker is at the left (right, respectively) position, we regard that the RLEM is in the state 0 (1).

First, consider the case where it is in the state 0 and a glider is given to the input port b as in Fig. 14. The glider first turns right at the upper right blinker. Then, at P it shifts the marker to the position of the right circle, and it turns right. By this, the state changes from 0 to 1. Finally, it further turns right at the upper left blinker, and goes out from the output port x .

Next, consider the case where it is in the state 0 and a glider is given to the input port a . The glider first turns right at the lower left blinker. Then it goes straight at Q without shifting the

position marker. It means that the glider knows that the state is 0. It further makes four right-turns, and finally goes out from the output port x . It should be noted that at P two different signal paths of the two cases are merged into one. When we want to reversibly merge two paths into one, knowing the state is necessary.

The remaining two cases are similar, and thus we can see the pattern correctly simulates RLEM 2-2. Note that in the case of RLEM 2-2, only two operations, which are the right-turn of a glider (Fig. 5) and the shifting of a blinker (Fig. 7), are used.

RLEM 2-3

RLEM 2-3 is simulated by the pattern shown in Fig. 15. As in the case of RLEM 2-2 (Fig. 14), a blinker is put near the center of the pattern as a position marker to keep the state of RLEM 2-3.

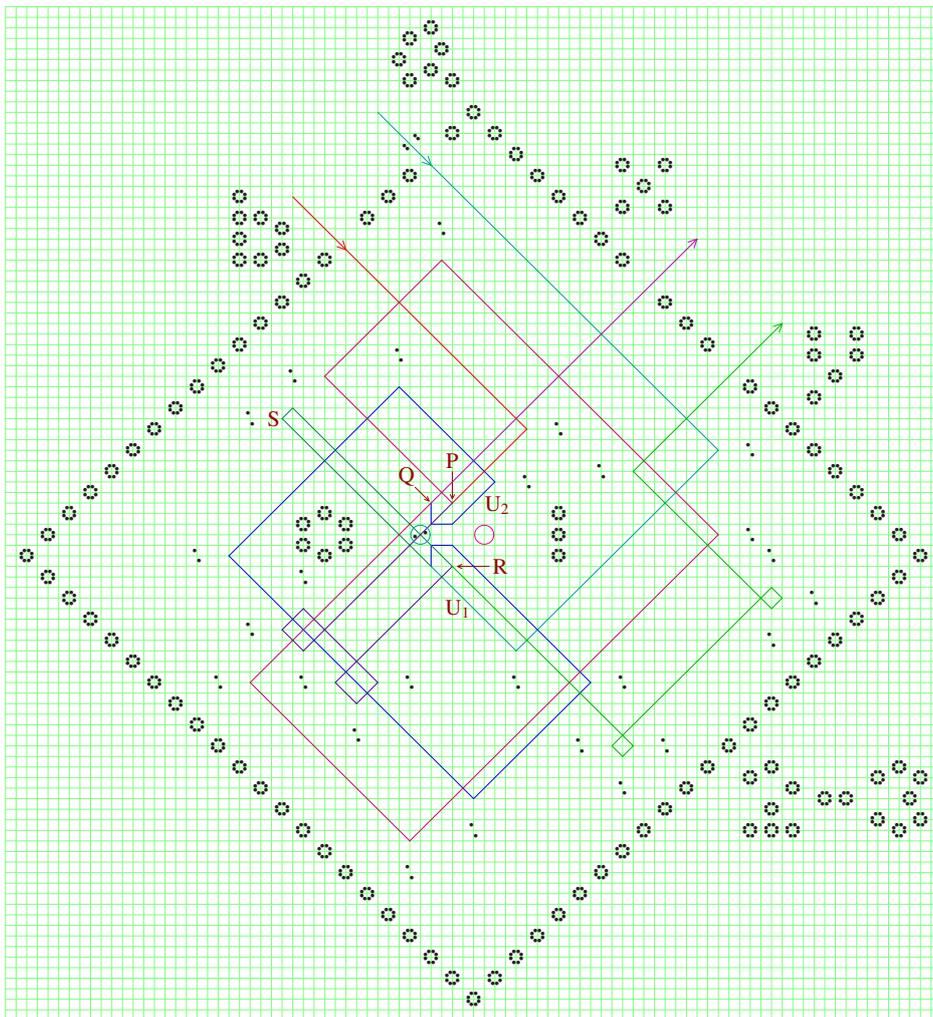


Figure 15: RLEM 2-3 implemented in P_0

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

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

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

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

RLEM 2-4

It is easy to obtain a pattern that simulates RLEM 2-4 from the pattern for RLEM 2-3. This is because the move function of RLEM 2-4 is the inverse of that of RLEM 2-3. The move function of RLEM 2-3 is as follows.

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

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

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

In [5], it is shown that in a reversible triangular partitioned CA (ETPCA), a pattern for “a backward functional module” can be easily obtained from an original pattern by a simple transformation. A similar property holds for reversible ESPCAs (though its details are omitted here). By this, the pattern that simulates RLEM 2-4 is obtained by putting blinkers at the mirror image positions of blinkers of the patterns of RLEM 2-3. It is shown in Fig. 16.

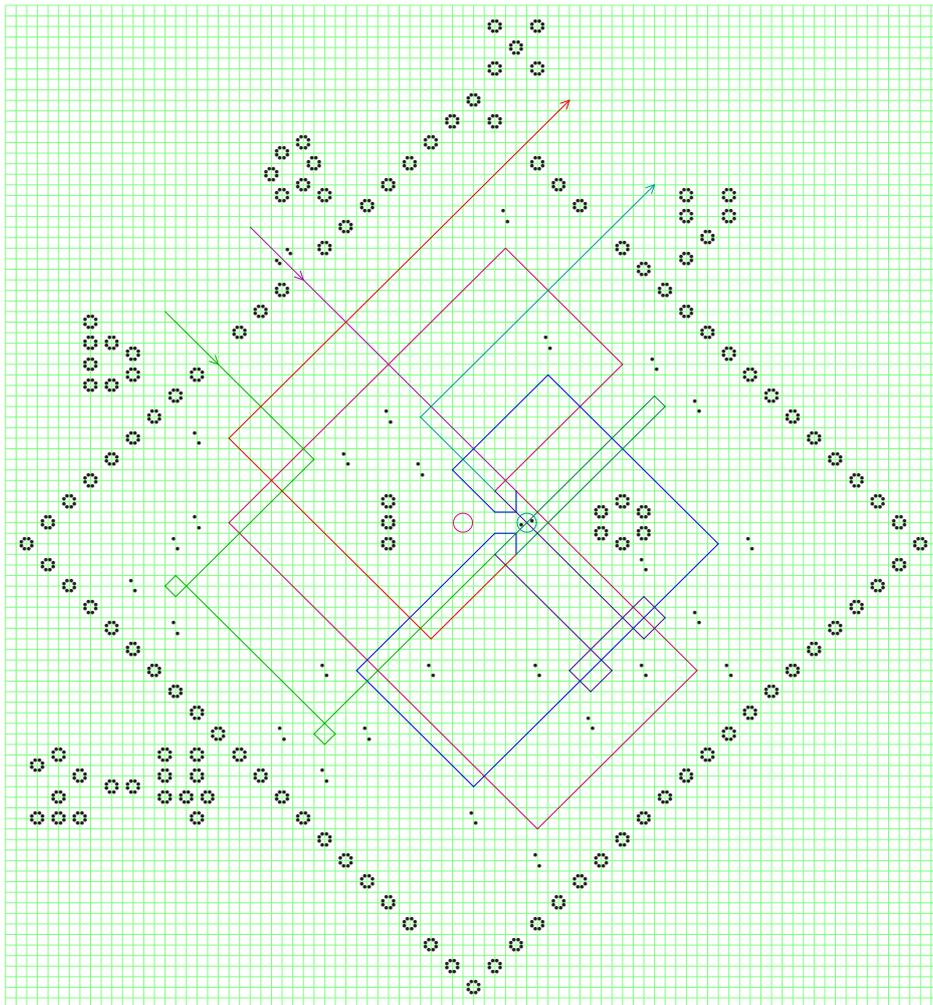


Figure 16: RLEM 2-4 implemented in P_0

2.3.2 RLEM 3-10

Combining the patterns for RLEMs 2-3 and 2-4 (Figs. 15 and 16) to form the circuit shown in Fig. 11, we obtain a pattern that simulates RLEM 3-10 (Fig. 17).

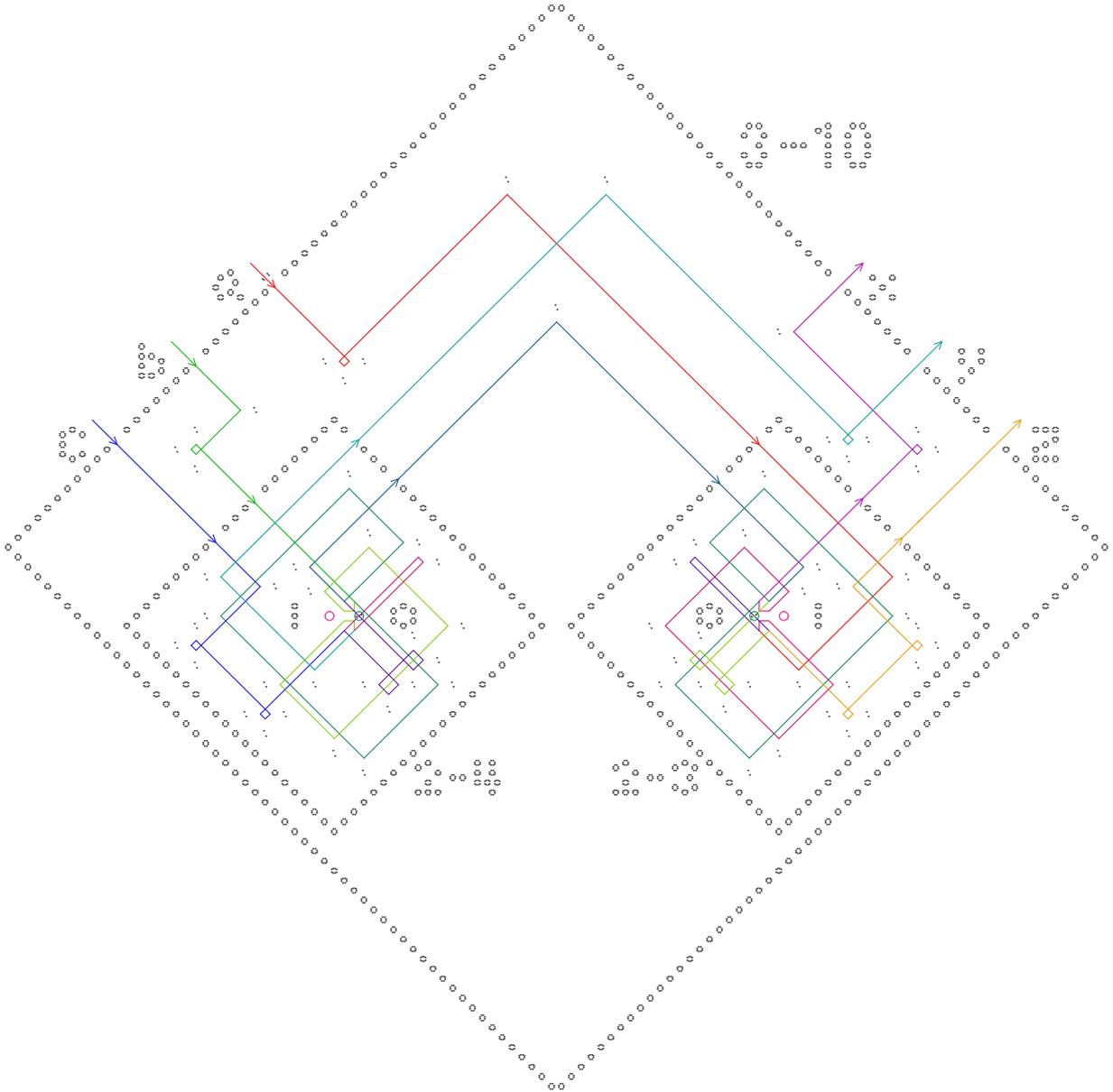


Figure 17: RLEM 3-10 implemented in P_0

2.3.3 RE

Putting eight copies of the pattern for RLEM 3-10 (Fig. 17) and connecting them to form the circuit shown in Fig. 10, we can obtain a pattern for RE. However, since it is very large ($2,000 \times 2,000$), its figure is omitted here (see the pattern file `08_RE.by_3-10.rle` using *Golly*).

2.3.4 RLEM 4-31

A pattern that simulates RLEM 4-31 is given in Fig. 18. We use three markers to keep the state 0 or 1, and thus there are three pairs of small circles around the center of Fig. 18, which show possible positions of the markers. If the three markers are all placed in the left (right, respectively) circles of the three pairs, we assume that the state is 0 (1). Since there are eight cases of the combination of states and symbols, we must prepare a sufficient number of access paths leading to the markers to perform operations of testing the state, changing the state, and merging signal paths. By this reason, we need three position markers. In this construction, we also use the U-turn operation (Fig. 6) for testing the state and merging signal paths.

Figure 18 shows the case where the state is 0 and a glider is given to the input port d . In this case, the glider first collides with the right position marker, and shifts it to the right. Then the glider shifts the central and the left position markers to the right. By this, the state changes from 0 to 1. Finally, the glider goes out from the output port w .

Next, consider the case where the state is 0 and a glider is given to the input port a . The glider tests whether the state is 1 at the left position marker. But, since the state is 0 in this case, the glider moves straight ahead at this position, and finally goes out from the output port w . Merging the two paths into one, which leads to w , is performed at the left position marker.

Since other cases are similar, the pattern shown in Fig. 18 correctly simulates RLEM 4-31.

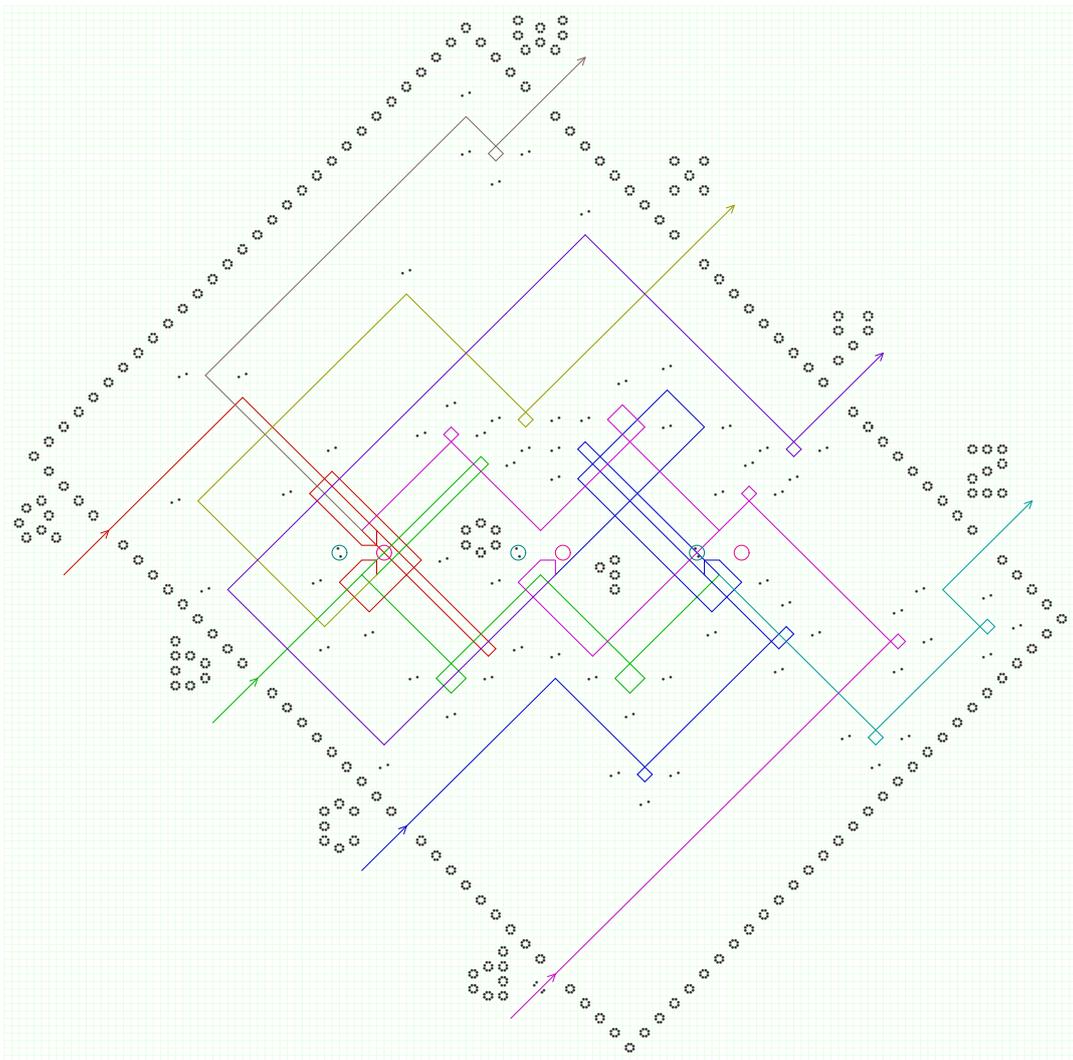


Figure 18: RLEM 4-31 implemented in P_0

3 Reversible Turing machines implemented in ESPCA P_0

A *reversible Turing machine* (RTM) is a deterministic TM that is also deterministic to the backward time direction (see e.g., [3] for the detailed definitions).

Consider an RTM T_{parity} that has the set of quintuples.

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

For example, $[q_0, 0, 1, R, q_1]$ means that if T_{parity} reads the symbol 0 in the state q_0 , then rewrite the symbol to 1, shift the head to the right, and go to the state q_1 . Assume a symbol string 01^n0 ($n = 0, 1, \dots$) is given. Then, T_{parity} halts in the accepting state q_{acc} if and only if n is even, and all the read symbols are complemented. Fig. 19 shows the computing process for the input string 0110.

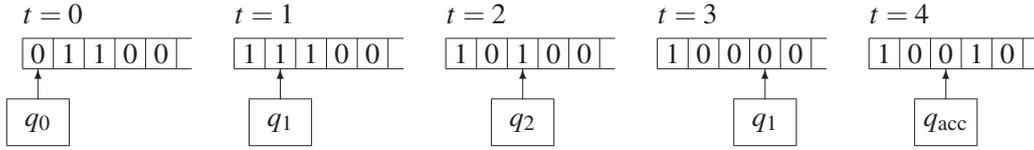


Figure 19: A computing process of the RTM T_{parity} for the given unary number 2

The second example is an RTM T_{power} with the following set of quintuples.

$$\{[q_0, 0, 0, R, q_1], [q_1, 0, 0, R, q_2], [q_2, 0, 0, L, q_6], [q_2, 1, 0, R, q_3], [q_3, 0, 1, L, q_4], [q_3, 1, 1, R, q_3], [q_4, 0, 0, L, q_7], [q_4, 1, 0, L, q_5], [q_5, 0, 1, R, q_2], [q_5, 1, 1, L, q_5], [q_6, 0, 0, L, q_{\text{rej}}], [q_6, 1, 1, R, q_1], [q_7, 0, 0, L, q_{\text{acc}}], [q_7, 1, 1, L, q_{\text{rej}}]\}$$

Assume a symbol string 001^n0 ($n = 0, 1, \dots$) is given as an input. Then, T_{power} halts in the accepting state q_{acc} if and only if $n = 2^k$ holds for some $k \in \{0, 1, 2, \dots\}$ (Fig. 20).

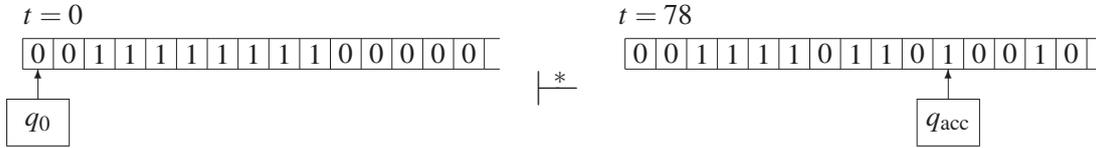


Figure 20: A computing process of the RTM T_{power} for the given unary number 8

In the following, the above examples of RTMs are constructed in P_0 .

3.1 Implementing RTMs in ESPCA P_0 using RE

It has been shown that any RTM can be constructed out of REs concisely [2, 3]. Figure 21 gives the whole circuit for simulating T_{parity} for the unary input 1. The circuit consists of two components. They are a finite control unit (left), and a tape unit (right). The tape unit is composed of an infinite copies of a tape cell module, which contains ten REs. Nine REs among ten form a column, and the remaining one is placed to the right of the column. Each tape cell simulates one square of the tape. The right RE of a tape cell keeps a tape symbol. The left RE column executes read/write and head-shift commands sent from the finite control. The bottom RE of the column keeps the head position. If the states of the bottom RE is H, then the head position is at this tape cell, else the head is not here. The finite control unit consists of state modules that generate read/write and shift commands. If a particle is given to the “Start” port, it starts to compute. Its answer will be obtained at “Accept” or “Reject” port.

Putting copies of the pattern of an RE in ESPCA P_0 at the corresponding position of the REs in Fig. 21, we obtain a configuration of P_0 that simulates the RTM T_{parity} . Fig. 22 is the configuration that simulates the RTM T_{power} in P_0 .

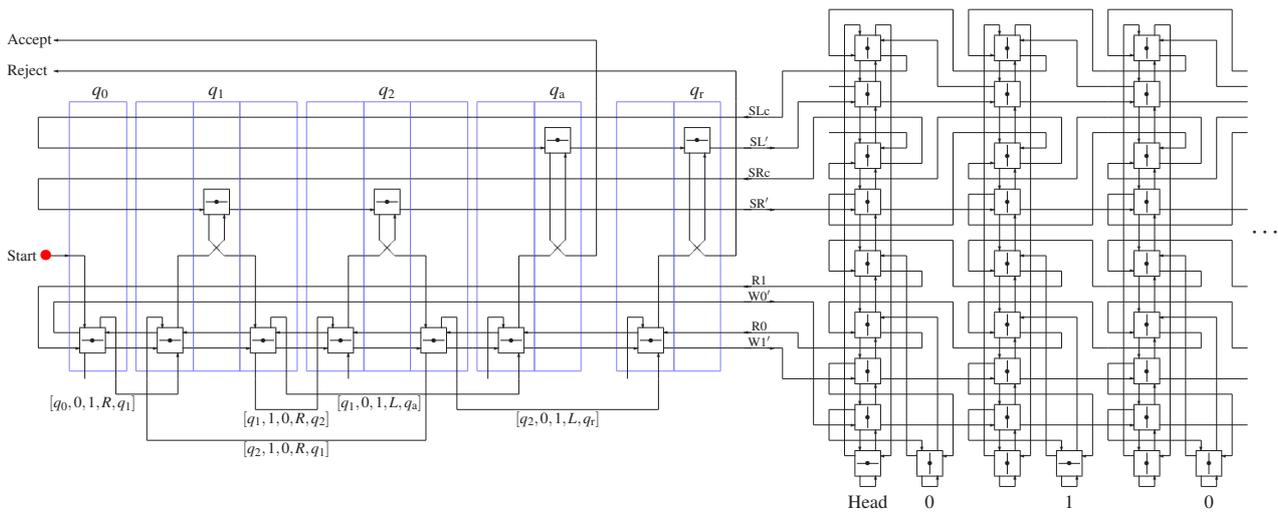


Figure 21: RTM T_{parity} composed of RE [2, 3]

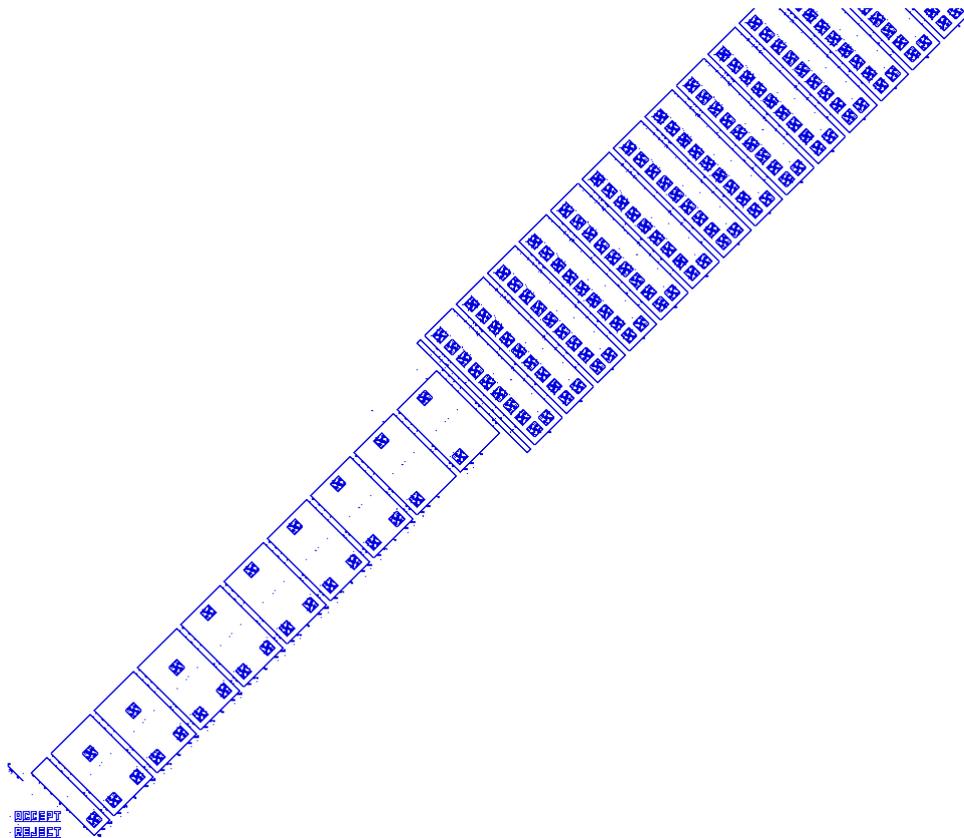


Figure 22: RTM T_{power} composed of RE implemented in P_0 simulated on *Golly*

3.2 Implementing RTMs in ESPCA P_0 using RLEM 4-31

It is also known that any RTM can be constructed out of RLEM 4-31 [7]. Fig. 23 gives the whole circuit for simulating T_{parity} for the input 1. Again the circuit consists of a finite control unit (left), and a tape unit (right). The tape unit is composed of an infinite copies of a tape cell consisting of nine RLEMs. The top RLEM of a tape cell keeps a tape symbol. The remaining eight RLEMs execute read/write and head-shift commands. They also keep the head position. If the states of the eight RLEMs are all 1, then the head position is at this tape cell. If they are all 0, the head is not here. If a particle is given to the “Start” port, it starts to compute. Its answer will be obtained at “Accept” or “Reject” port.

Putting copies of the pattern of RLEM 4-31 given in Fig. 18 at the positions corresponding to the RLEMs in Fig. 23, and connecting them appropriately, we have a complete configuration of ESPCA P_0 that simulates T_{parity} . Figure 24 shows a configuration of P_0 simulated on *Golly* that realizes the circuit for T_{power} .

4 Pattern files for *Golly*

The zipped file `ESPCA_01caef.zip` contains emulator files (`*.rule`) and pattern files (`*.rle`) for *Golly* [9]. Putting `ESPCA_01caef.zip` in the “Patterns” folder of *Golly*, and accessing `*.rle` files from *Golly*, evolving processes of configurations of P_0 can be seen. In particular, whole computing processes of RTMs are observed. The pattern files given here are as follows.

- `01_Glider_and_blinkers.rle`
It shows how the three operations given in Figs. 5–7 work.
- `02_RLEM_2-2.rle`
It simulates RLEM 2-2 shown in Fig. 14.
- `03_RLEM_4-31.rle`
It simulates RLEM 4-31 shown in Fig. 18.
- `04_RTM_parity_by_4-31.rle`
It simulates RTM T_{parity} made of RLEM 4-31 shown in Fig. 23 for the inputs $n = 2$ and 3.
- `05_RTM_power_by_4-31.rle`
It simulates RTM T_{power} made of RLEM 4-31 shown in Fig. 24 for the inputs $n = 4, 6$ and 8.
- `06_RLEMs_2-3_and_2-4.rle`
It simulates RLEMs 2-3 and 2-4 shown in Figs. 15 and 16.
- `07_RLEM_3-10_by_2-3_and_2-4.rle`
It simulates RLEM 3-10 shown in Fig. 17.
- `08_RE_by_3-10.rle`
It simulates RE composed of RLEM 3-10 shown in Fig. 10.
- `09_RTM_parity_by_RE.rle`
It simulates RTM T_{parity} made of RE shown in Fig. 21 for the inputs $n = 2$ and 3.
- `10_RTM_power_by_RE.rle`
It simulates RTM T_{power} made of RE shown in Fig. 22 for the input $n = 4$.

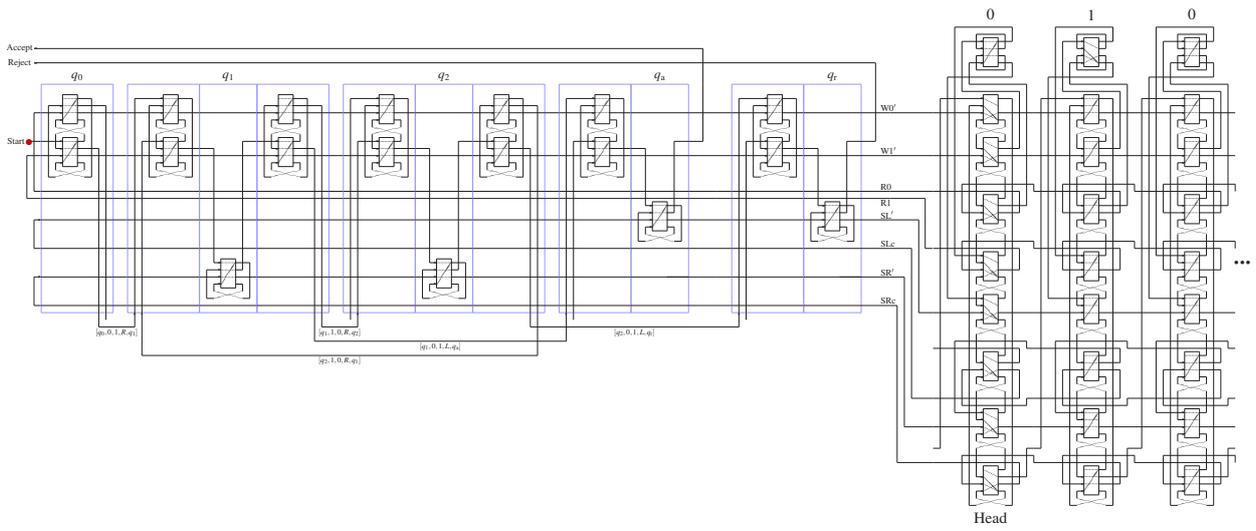


Figure 23: RTM T_{parity} composed of RLEM 4-31 [7]

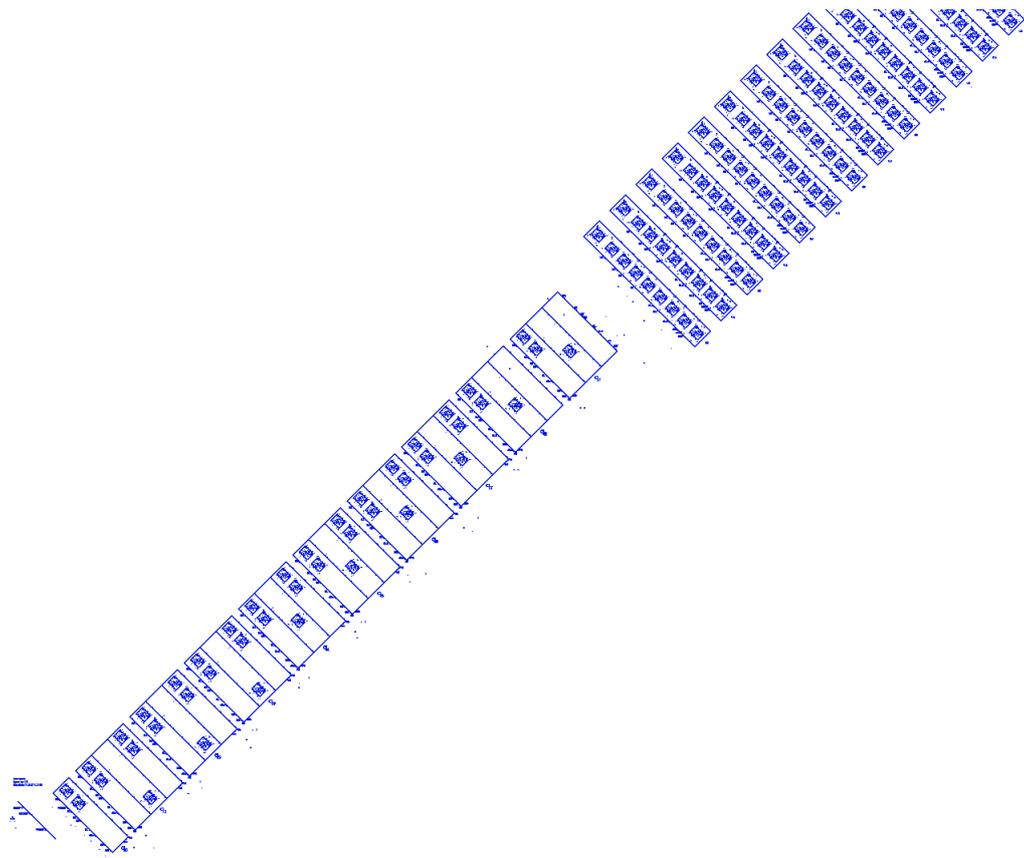


Figure 24: RTM T_{power} composed of RLEM 4-31 implemented in P_0 simulated on *Golly*

Note: A part of this report is presented at [4].

Revision history: In the first version (December 2021), the pattern files 01–05 are given, and a construction method of patterns for RTMs using RLEM 4-31 is shown. In the second version (March 2022), the pattern files 06–10 are added, and a construction method of RTMs using RE is shown.

Acknowledgements: I express my gratitude to the developing and support teams of *Golly* [9].

References

- [1] 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](https://doi.org/10.1007/978-3-540-79992-4_9)
- [2] 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](https://doi.org/10.1007/3-540-45132-3_6)
- [3] Morita, K.: Theory of Reversible Computing. Springer, Tokyo (2017). doi:[10.1007/978-4-431-56606-9](https://doi.org/10.1007/978-4-431-56606-9)
- [4] Morita, K.: Computing in a simple reversible and conservative cellular automaton. In: Proc. First Asian Symposium on Cellular Automata Technology (eds. S. Das, G.J. Martinez) AISC 1425 (in press). doi:[10.1007/978-981-19-0542-1_1](https://doi.org/10.1007/978-981-19-0542-1_1)
- [5] Morita, K.: Gliders in the Game of Life and in a reversible cellular automaton. In: The Mathematical Artist – A Tribute To John Horton Conway (eds. S. Das, S. Roy, K. Bhattacharjee). Springer (in press). doi:[10.1007](https://doi.org/10.1007)
- [6] 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)
- [7] Morita, K., Suyama, R.: Compact realization of reversible Turing machines by 2-state reversible logic elements. In: Proc. UCNC 2014 (eds. O.H. Ibarra, L. Kari, S. Kopecki), LNCS 8553, pp. 280–292 (2014). doi:[10.1007/978-3-319-08123-6_23](https://doi.org/10.1007/978-3-319-08123-6_23)
- [8] Mukai, Y., Ogiro, T., Morita, K.: Universality problems on reversible logic elements with 1-bit memory. *Int. J. Unconventional Computing* **10**, 353–373 (2014)
- [9] Trevorrow, A., Rokicki, T., Hutton, T., et al.: Golly: an open source, cross-platform application for exploring Conway’s Game of Life and other cellular automata. <http://golly.sourceforge.net/> (2005)