Simulating a reversible elementary square partitioned cellular automaton with the ID number 01caef on Golly

Kenichi Morita*

December 2021

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. Examples of whole computing processes of RTMs can be seen on *Golly* using the files given here.

Elementary square partitioned cellular automaton (ESPCA)

A 4-neighbor *square partitioned cellular automaton* (SPCA) is a two-dimensional CA whose 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)**.

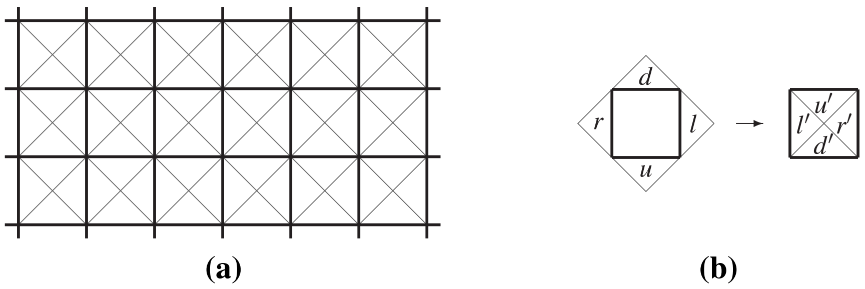

Figure 1: **(a)** Cellular space of a 4-neighbor square partitioned cellular automaton (SPCA), and **(b)** its local transition rule

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.

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., [1] 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.

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

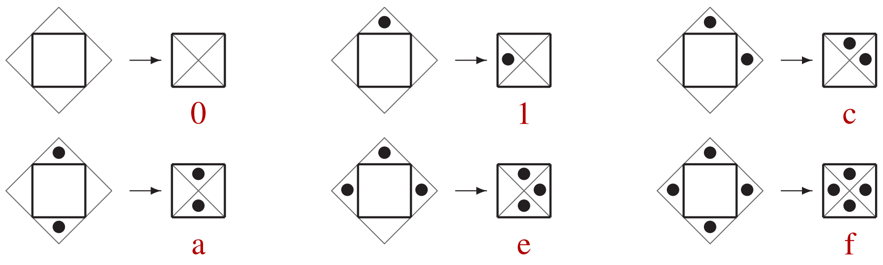

Figure 2: Local transition function defined by the six local transition rules of a particular reversible and conservative ESPCA 01caef, which is denoted by P_0 hereafter

Useful patterns in ESPCA P_0

The pattern shown in Fig. 3 is called a *glider* that flies one cell diagonally in 12 steps.

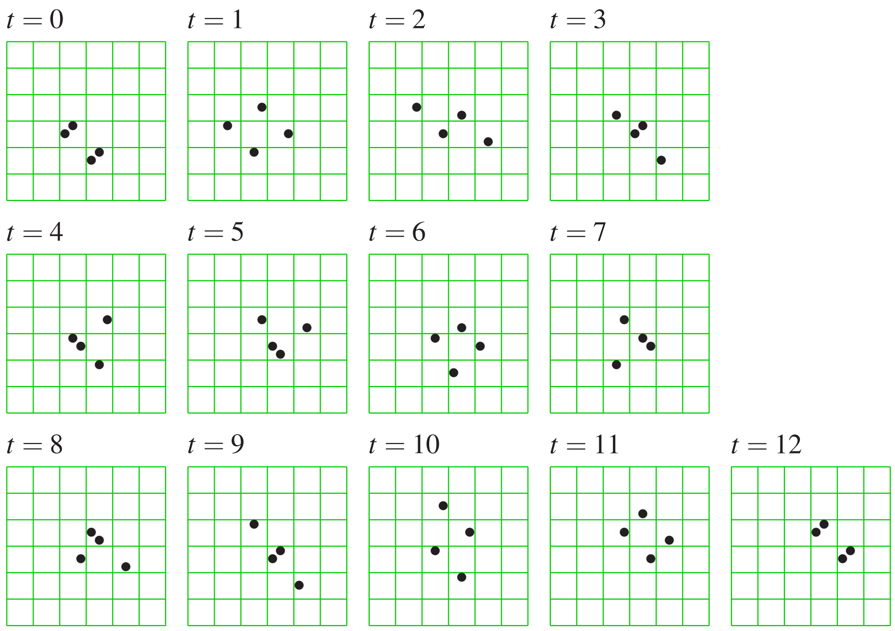

Figure 3: Glider in P_0

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 . The latter will be used only for writing comments and indicating a border of a logic element in the cellular space.

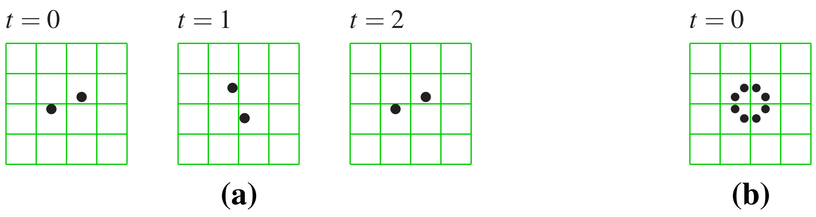

Figure 4: Periodic patterns in P_0. **(a)** Blinker, and **(b)** block

Three useful phenomena in ESPCA P_0

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

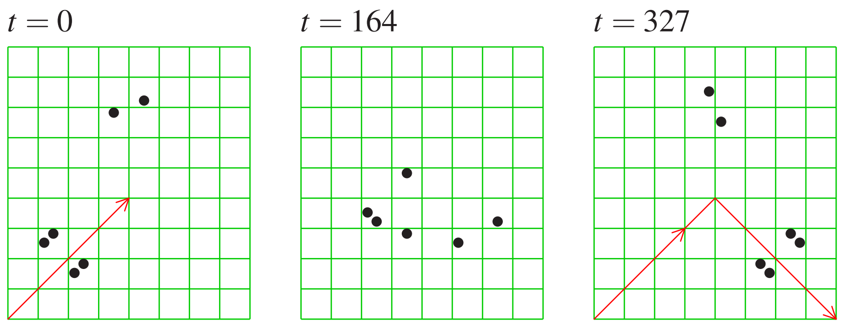

Figure 5: Right-turn of a glider in P_0

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.

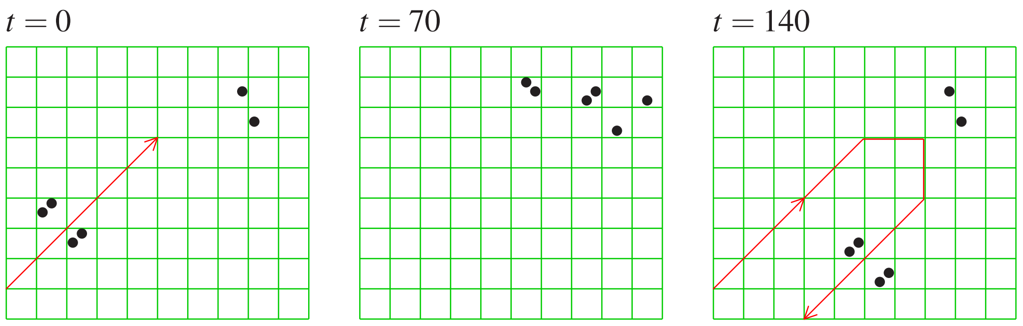

Figure 6: U-turn of a glider in P_0

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.

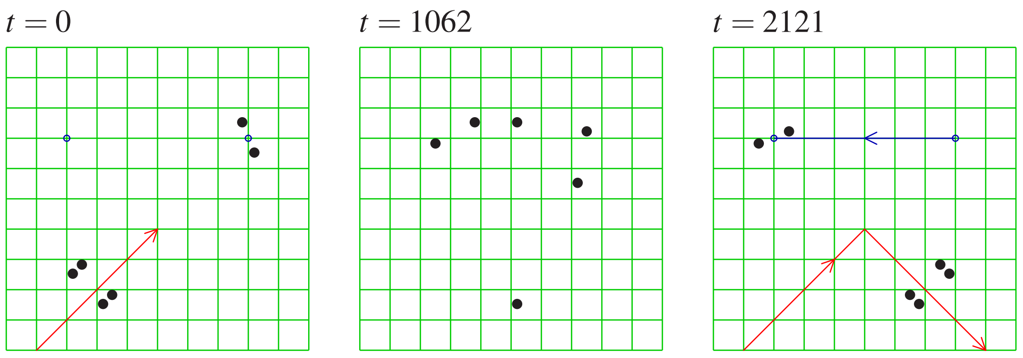

Figure 7: Shifting a blinker by a glider in P_0

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) [1, 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. It is known that every 2-state RLEM that has three or more symbols is *universal*, which means any RLEM is composed only of it [1]. In the following we use the 2-state 4-symbol universal RLEM No. 4-31 for composing RTMs.

The move function of RLEM 4-31 is represented in a graphical form as in Fig. 8. Two rectangles in the figure correspond to the two states 0 and 1. Solid and dotted lines show the input-output relation. If an input signal goes through a dotted line, then the state does not change (Fig. 9 **(a)**). If a signal goes through a solid line, then the state changes (Fig. 9 **(b)**).

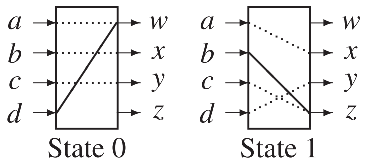

Figure 8: A four-symbol two-state RLEM 4-31

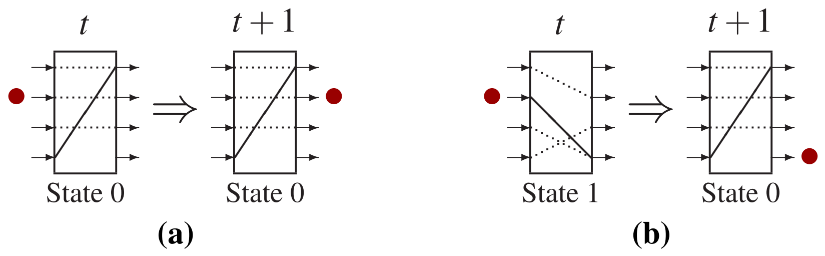

Figure 9: Operations of RLEM 4-31. **(a)** A case where a signal goes through a dotted line, and **(b)** a case where it goes through a solid line

We first show that RLEM 2-2 (Fig. 10) can be realized in the cellular space of ESPCA P_0. Although RLEM 2-2 is not universal [1], it is implemented rather simply. Therefore, it is convenient for explaining how we can design a pattern that simulates an RLEM in P_0.

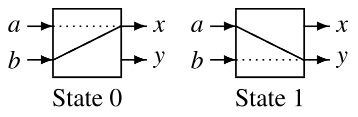

Figure 10: A two-symbol two-state RLEM 2-2

Figure 11 shows a pattern that simulates RLEM 2-2. 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. 11. 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 ahead 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.

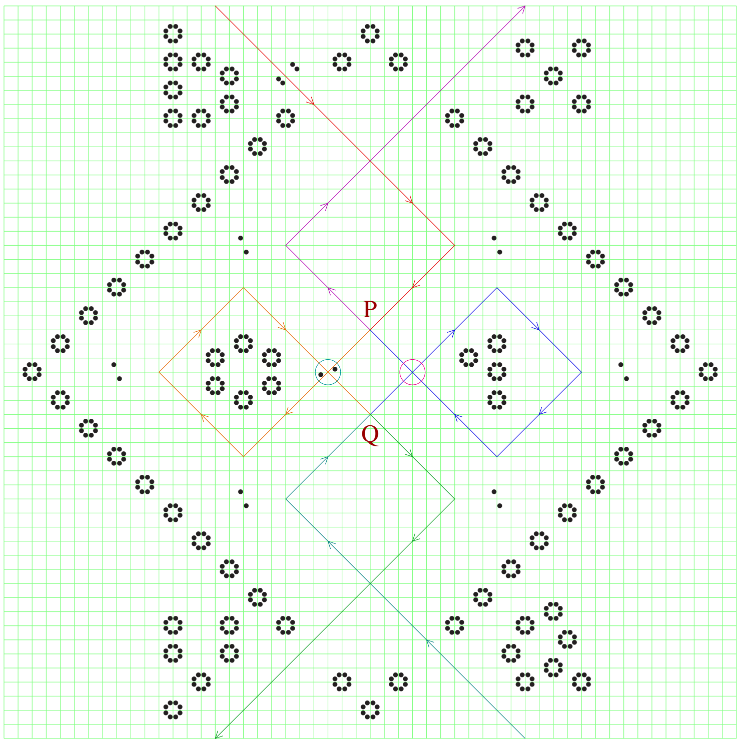

Figure 11: RLEM 2-2 implemented in P_0

A pattern that simulates RLEM 4-31 is given in Fig. 12. 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. 12, 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 12 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, we can see that the pattern shown in Fig. 12 correctly simulates RLEM 4-31.

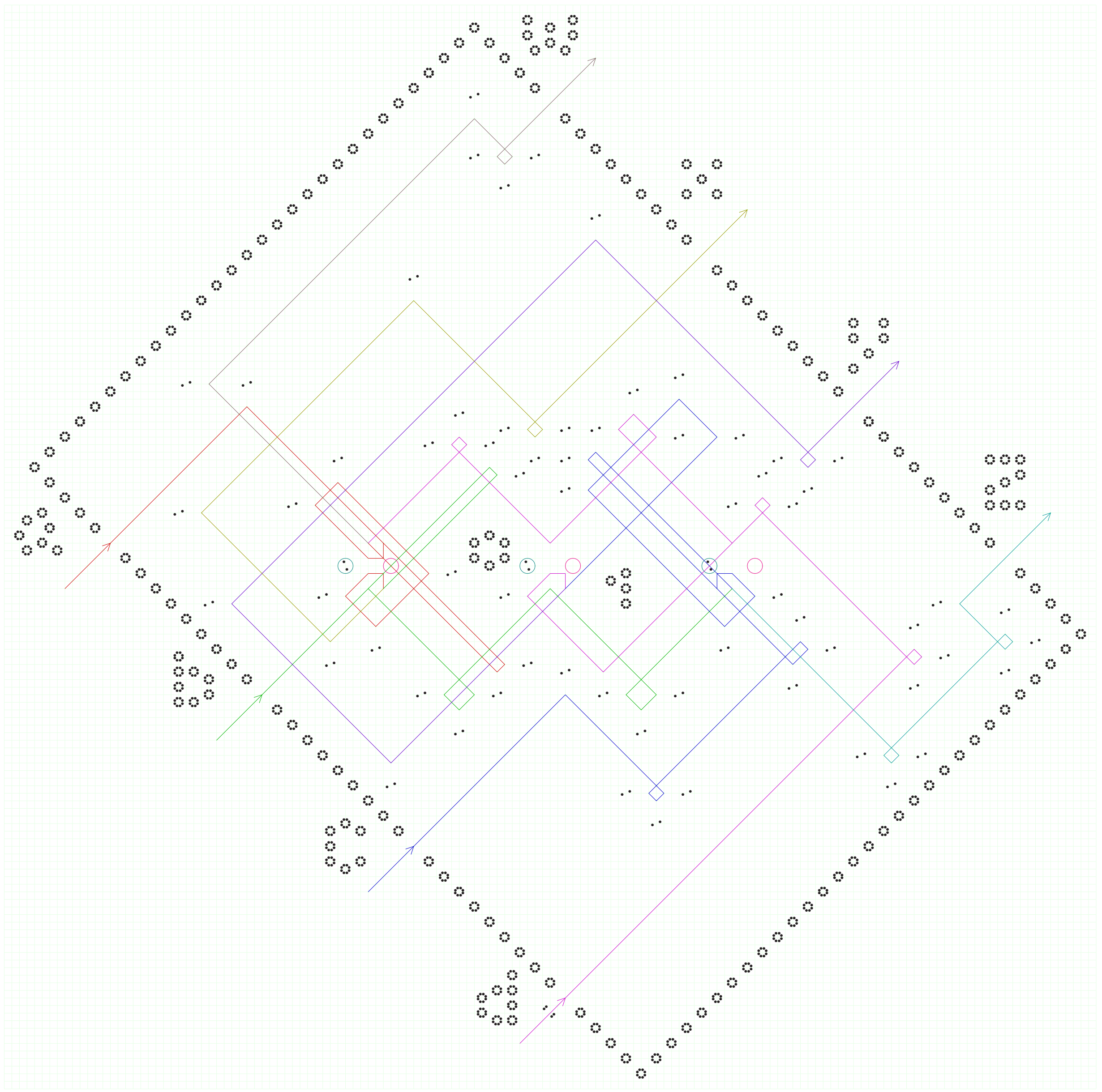

Figure 12: RLEM 4-31 implemented in P_0

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., [1] 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 $0\,1^n\,0$ $(n = 0,1,\dots)$ 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. Figure 13 shows the computing process for the input string 0110.

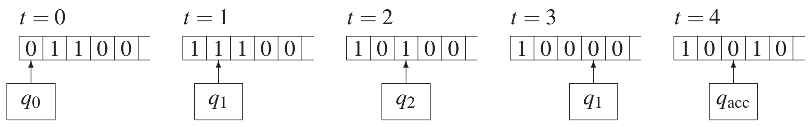

Figure 13: A computing process of the RTM $T_{\rm parity}$ for the given unary number 2

It has been shown that any RTM can be constructed out of RLEM 4-31 concisely [3]. Figure 14 gives the whole circuit for simulating $T_{\rm parity}$ for the 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 memory cell module, which is a vertical array of nine RLEMs. Each memory cell simulates one square of the tape. The top RLEM of a memory cell keeps a tape symbol. The remaining eight RLEMs execute read/write and head-shift commands sent from the finite control. They also keep the head position. If the states of the eight RLEMs are all 1, then the head position is at this memory cell. If they are all 0, the head is not here. If a particle is given to the "Begin" port, it starts to compute. Its answer will be obtained at "Accept" or "Reject" port.

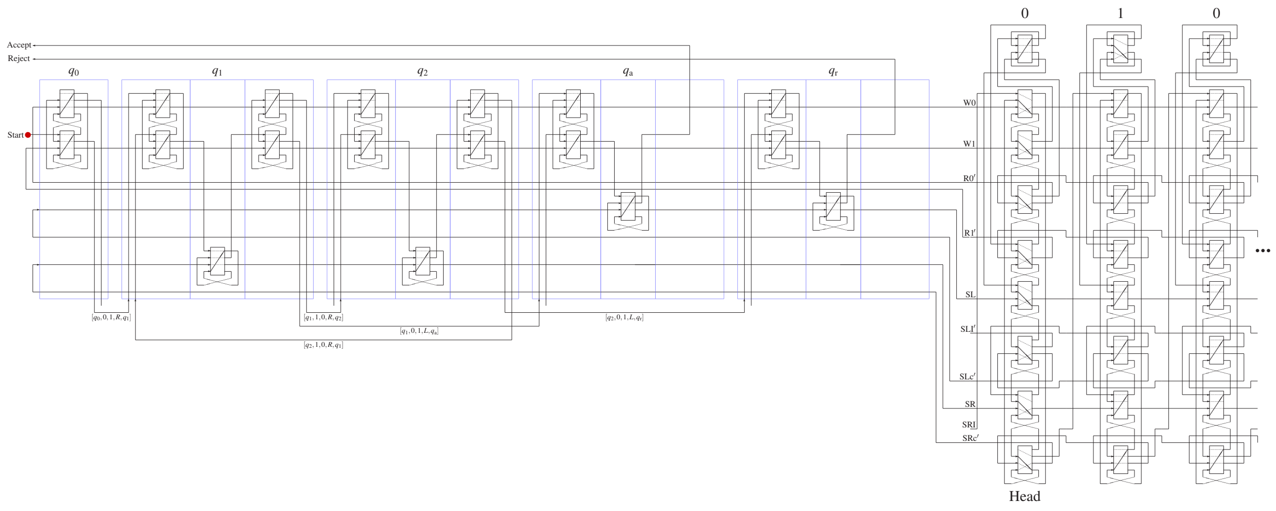

Figure 14: RTM $T_{\rm parity}$ composed of RLEM 4-31 [3]

Putting copies of the pattern of RLEM 4-31 given in Fig. 12 at the positions corresponding to the RLEMs in Fig. 14, and connecting them appropriately, we have a complete configuration of ESPCA P_0 that simulates $T_{\rm parity}$.

Consider another RTM $T_{\rm power}$ with the following set of quintuples.

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

Assume a symbol string $0\,0\,1^n\,0$ $(n = 0,1,\ldots)$ is given as an input. Then, $T_{\rm power}$ halts in the accepting state $q_{\rm acc}$ if and only if $n = 2^k$ holds for some $k \in \{0,1,2,\ldots\}$. Figure 15 shows the computing process for the input string $0\,0\,1^8\,0$. A circuit for $T_{\rm power}$ is also constructed out of RLEM 4-31. Figure 16 shows a configuration of P_0 simulated on *Golly* that realizes the circuit for $T_{\rm power}$.

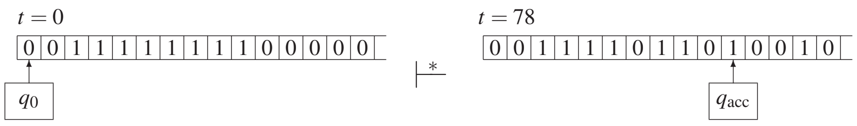

Figure 15: A computing process of the RTM $T_{\rm power}$ for the given unary number 8

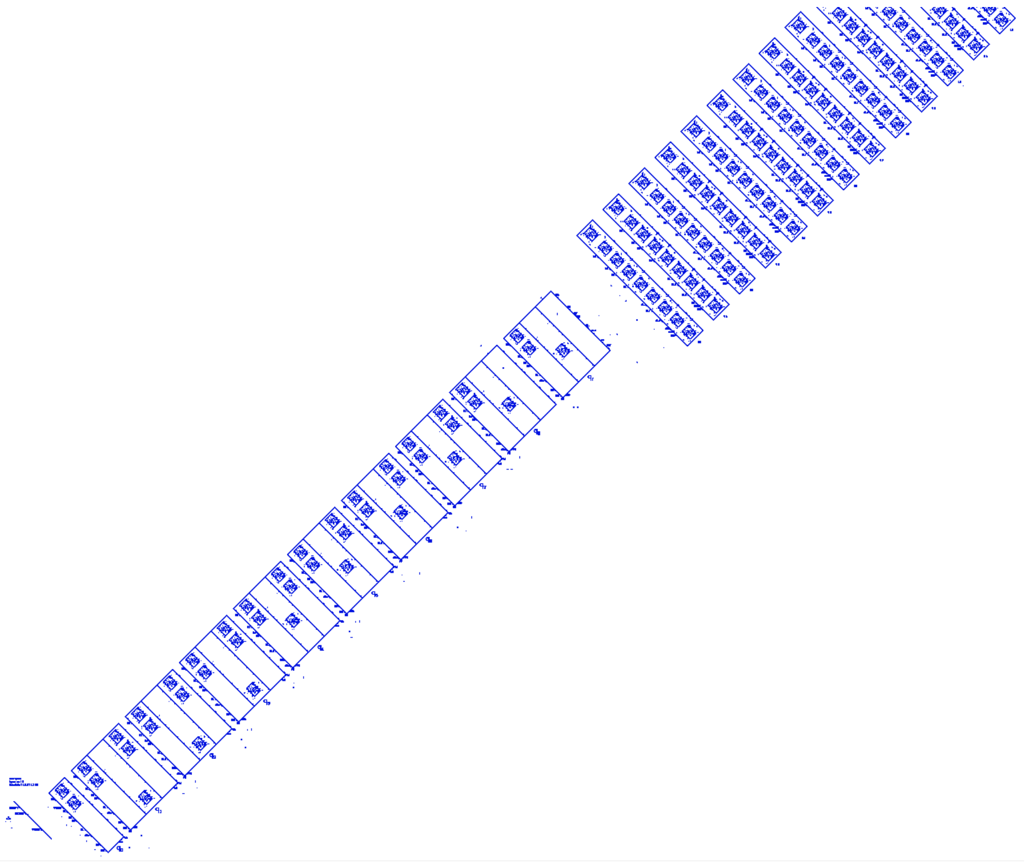

Figure 16: RTM T_{power} in P_0 simulated on *Golly*

Pattern files for *Golly*

The zipped file `ESPCA_01caef.zip` contains emulator files (`*.rule`) and pattern files (`*.rle`) for *Golly* [4]. 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.

- 1_glider_and_blinkers.rle
  It shows how the three operations given in Figs. 5–7 work.
- 2_RLEM_2-2.rle
  It simulates RLEM 2-2 shown in Fig. 11.
- 3_RLEM_4-31.rle
  It simulates RLEM 4-31 shown in Fig. 12.
- 4_RTM_parity.rle
  It simulates RTM T_{parity} shown in Fig. 14 for the inputs $n = 2$ and 3.
- 5_RTM_power.rle
  It simulates RTM T_{power} shown in Fig. 16 for the inputs $n = 4$, 6 and 8.

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

Note: The contents of this report will be presented at [2].

References

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

[2] Morita, K.: Computing in a simple reversible and conservative cellular automaton. In: Proc. First Asian Symposium on Cellular Automata Technology (ASCAT 2022) (to appear)

[3] 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

[4] 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)