Theoretical Computer Science 168 (1996) 337–366

Theoretical Computer Science

Self-reproduction in a reversible cellular space

Kenichi Morita*, Katsunobu Imai

Department of Industrial and Systems Engineering, Faculty of Engineering, Hiroshima University, Higashi-Hiroshima-shi, 739 Japan

Abstract

We investigate a problem whether self-reproduction is possible in a two-dimensional "reversible" cellular space, and give an affirmative answer. A reversible (or injective) cellular automaton (RCA) is a CA such that every configuration has at most one predecessor. In order to design an RCA we use a framework of partitioned cellular automaton (PCA). A PCA with von Neumann neighborhood is a special type of CA whose cell is divided into five parts. We designed here a reversible PCA SR_8 having 8 states in each part (thus one cell has 8^5 states). In this cellular space, encoding the shape of an object into a "gene" represented by a command sequence, copying the gene, and interpreting the gene to create an object, are all performed reversibly. We show that, by using these operations, various objects can reproduce themselves in a very simple manner.

1. Introduction

It is well known that von Neumann [9] first showed a computation- and construction-universal self-reproducing machine using his 29-state two-dimensional cellular automaton. Since then, various models of cellular automata that support self-reproduction have been studied. For example, Codd [2] proposed an 8-state cellular model that has essentially the same ability as von Neumann's model. Langton [3] showed that, if universal computing and construction abilities are not required, a very simple self-reproducing object can be designed in a modified 8-state model of Codd.

We investigate here a problem whether self-reproduction is possible in a two-dimensional "reversible" cellular space. A reversible (or injective) cellular automaton (RCA) is a CA such that every configuration has at most one predecessor. It can be regarded as a model of reversible physical space. In spite of the constraint of reversibility, RCA has rich ability of information processing. Toffoli [10] showed that every (irreversible) k-dimensional CA can be simulated by a $k+1$-dimensional RCA in real time. Another method to simulate an irreversible CA by an RCA without increasing the dimension

* Corresponding author. E-mail: {morita, imai}@ke.sys.hiroshima-u.ac.jp.

(but not in real time) was proposed by Morita [6]. From these results, existence of computation- and construction universal (and thus self-reproducing) RCA can be concluded. However, if we use these "emulation methods" to convert an irreversible CA to an RCA, a large amount of garbage signals are generated and spread out in the reversible cellular space. Therefore, direct and elegant method for designing RCA having desired property should be explored.

As for computation-universal RCA, Margolus [4] proposed an elegant model of 2-state RCA with so called Margolus neighborhood. Morita and Ueno [8] showed two models of computation-universal 16-state RCA having the usual von Neumann neighborhood. It has also been shown that a reversible Turing machine, which is known to be universal [1], can be directly embedded in a one-dimensional RCA [7, 5]. However, until now no direct method to construct a self-reproducing RCA has been known.

We give here an RCA in which various objects can self-reproduce in a very simple fashion, although they have neither computation- nor construction-universality in von Neumann's sense. However, we do not consider this type of self-reproduction as a trivial self-replication of patterns. Because these objects can handle with a "command sequence" or a "gene", and have an ability of encoding, decoding and executing it. Self-reproduction is carried out by interpreting a gene as a kind of program.

In order to design an RCA we use a framework of partitioned cellular automaton (PCA) [7]. A PCA with von Neumann neighborhood is a special type of CA whose cell is divided into five parts. The next state of each cell is determined by the present states of the center part of this cell, the lower part of the upper cell, the left part of the right cell, the upper part of the lower cell, and the right part of the left cell (not depending on the entire five cells). In PCA, injectivity of the global map is equivalent to injectivity of the local map [7]. Therefore, it makes easy to design an RCA.

We present a reversible PCA "SR_8" having 8 states in each of five parts. Thus one cell has 8^5 states. First, a signal transmission wire on which commands propagate is designed. We then give two types of objects called a "Worm" and a "Loop". A Worm is a simple wire with end points, while a Loop is a simple closed wire. We design the local transition function of SR_8 so that the operations of encoding the shape of a wire into a gene, copying the gene, and interpreting the gene to create a wire, are all performed reversibly in this cellular space. We show that *any* Worms and Loops satisfying some appropriate condition can reproduce themselves indefinitely using these operations.

2. Definitions and preliminaries

In this section, we give definitions of a (usual) cellular automaton (CA) and a partitioned cellular automaton (PCA), and then state some basic properties on PCA.

Definition 2.1. A *deterministic two-dimensional cellular automaton* (CA) with von Neumann neighborhood is a system defined by

$$A = (\mathbf{Z}^2, Q, f, \#),$$

where $\mathbf{Z}$ is the set of all integers, Q is a non-empty finite set of internal states of each cell, $f : Q^5 \to Q$ is a mapping called a *local function*, and $\# \in Q$ is a *quiescent state* that satisfies $f(\#,\#,\#,\#,\#) = \#$.

A *configuration* over Q is a mapping $\alpha : \mathbf{Z}^2 \to Q$. Let $\mathrm{Conf}(Q)$ denote the set of all configurations over Q, i.e., $\mathrm{Conf}(Q) = \{\alpha \mid \alpha : \mathbf{Z}^2 \to Q\}$.

The function $F : \mathrm{Conf}(Q) \to \mathrm{Conf}(Q)$ defined as follows is called the *global function* of A.

$$\forall (x,y) \in \mathbf{Z}^2 :$$

$$F(\alpha)(x,y) = f(\alpha(x,y), \alpha(x,y+1), \alpha(x+1,y), \alpha(x,y-1), \alpha(x-1,y))$$

We say A is a *reversible* (or *injective*) CA (denoted by RCA) iff F is one-to-one.

Definition 2.2. A *deterministic two-dimensional partitioned cellular automaton* (PCA) with von Neumann neighborhood is a system defined by

$$P = (\mathbf{Z}^2, (C, U, R, D, L), g, (\#,\#,\#,\#,\#)),$$

where C, U, R, D, and L are non-empty finite sets of states in center, up, right, down and left parts of each cell, $g : C \times D \times L \times U \times R \to C \times U \times R \times D \times L$ is a local function, and $(\#,\#,\#,\#,\#) \in C \times U \times R \times D \times L$ is a quiescent state satisfying $g(\#,\#,\#,\#,\#) = (\#,\#,\#,\#,\#)$.

Let `CENTER` (`UP`, `RIGHT`, `DOWN`, `LEFT`, respectively) be the projection function which picks out the center (up, right, down, left) element of a quintuple in $C \times U \times R \times D \times L$. The global function $G : \mathrm{Conf}(C \times U \times R \times D \times L) \to \mathrm{Conf}(C \times U \times R \times D \times L)$ of P is defined as follows.

$$\forall (x,y) \in \mathbf{Z}^2 :$$

$$\begin{aligned} G(\alpha)(x,y) = g(&\mathtt{CENTER}\,(\alpha(x,y)),\ \mathtt{DOWN}\,(\alpha(x,y+1)),\ \mathtt{LEFT}\,(\alpha(x+1,y)),\\ &\mathtt{UP}\,(\alpha(x,y-1)),\ \mathtt{RIGHT}\,(\alpha(x-1,y))) \end{aligned}$$

We say P is *globally reversible* iff G is one-to-one, and *locally reversible* iff g is one-to-one.

PCA is a subclass of CA such that each cell is partitioned into five parts (Fig. 1), and the next state of each cell is determined depending on the center part of this cell, the lower part of the upper cell, the left part of the right cell, the upper part of the lower cell, and the right part of the left cell.

State transition of a cell by the equation $g(c,d,l,u,r) = (c',u',r',d',l')$ can be depicted as in Fig. 2. We call this figure a *rule* of P. Besides such a figure, we also use an abbreviated figure as shown in Fig. 3 or the notation

$$[c,d,l,u,r] \ \to \ [c',u',r',d',l']$$

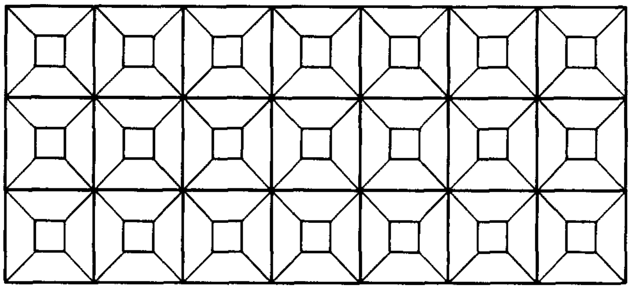

Fig. 1. Cellular space of PCA.

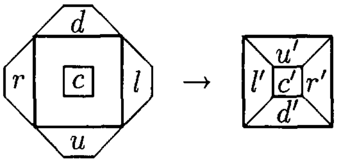

Fig. 2. A representation of a rule.

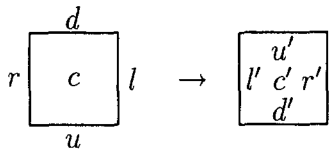

Fig. 3. An abbreviated representation of a rule.

to represent a rule of P. In what follows, we regard the local function g as the set of such rules.

It is easy to show the following propositions on PCA. Proofs are omitted here, since analogous results for one-dimensional PCA are proved in [7].

Proposition 2.1. *Let P be a PCA. P is globally reversible iff it is locally reversible.*

Proposition 2.2. *For any PCA P, there is a CA A whose global function is identical with that of P.*

By Proposition 2.1, globally or locally reversible PCA is called simply "reversible" and denoted by RPCA. Proposition 2.2 says that PCA is a subclass of CA.

By above, if we want to construct a reversible CA, it is sufficient to give a PCA whose local function g is one-to-one (note that the numbers of elements of domain and range of g are the same). As for a one-to-one mapping the following proposition holds (it is easily proved).

Proposition 2.3. *Let A and B be finite sets having the same number of elements. If $f' : A' \to B$ is one-to-one for $A' \subset A$, then there is a one-to-one mapping $f : A \to B$ that is an extension of f'.*

Thus, to give an RPCA, it is sufficient to define a one-to-one function g only on a subset of $C \times D \times L \times U \times R$ which is needed to show desired property.

3. Designing an RPCA that supports self-reproduction

We now give an RPCA "SR_8" that supports self-reproduction. It is defined by

$$SR_8 = (\mathbf{Z}^2, (C, U, R, D, L), g, (\#,\#,\#,\#,\#)),$$

$$C = U = R = D = L = \{\#, *, +, -, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}.$$

Hence, each of five parts of a cell has 8 states. The states A, B, C and D mainly act as signals that are used to compose "commands". The states $*, +$, and $-$ are used to control these signals.

The local function g is defined below. In SR_8, g is *isotropic*, i.e., invariant under the rotation of 90°, 180° and 270°. Namely, if there is a rule

$$[c,d,l,u,r] \rightarrow [c',u',r',d',l']$$

in g, then there are also the following three rules.

$$[c,r,d,l,u] \rightarrow [c',l',u',r',d']$$
$$[c,u,r,d,l] \rightarrow [c',d',l',u',r']$$
$$[c,l,u,r,d] \rightarrow [c',r',d',l',u']$$

Therefore, in this section, we write only one of the four rules and omit the other rotated ones. Rules in g and their functions are explained in order.

3.1. Quiescent state rule

This rule is shown in Fig. 4 (# is indicated as a blank).

3.2. Signal transmission on a wire

A *wire* is a configuration to transmit signals A, B, and C. Fig. 5 shows an example of a part of a simple (i.e., non-branching) wire.

By the rules (2)–(4) in Fig. 6, signals x_i and y_i in Fig. 5 shift by one cell along the wire in two time steps. Note that, strictly speaking, (2)–(4) are "rule schemes". For example, (2) represents 36 rules since $x, y \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ and it is isotropic.

A wire may contain two-way (bifurcating) or three-way (trifurcating) *branches*. A branching point acts as a "fan-out", i.e., the signals are copied at this point. Rules (5)–(7) in Fig. 7 are for two-way branch, and rule (8) is for three-way one. Fig. 8 shows an example of a wire with a bifurcation.

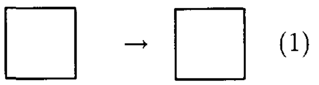

Fig. 4. Quiescent state rule.

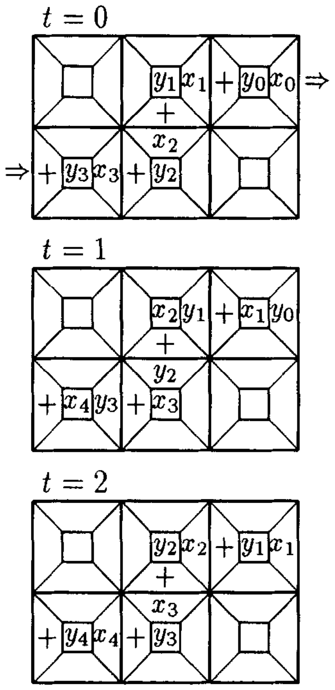

Fig. 5. Signal transmission on a part of a simple wire $(x_i, y_i \in \{A, B, C\})$.

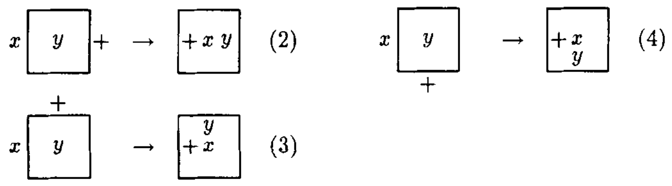

Fig. 6. Rules for signal transmission $(x \in \{A, B, C\},\ y \in \{A, B, C\})$.

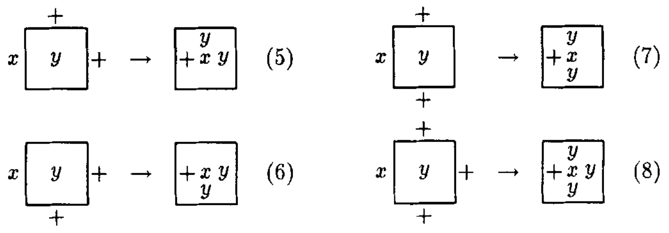

Fig. 7. Rules for signal transmission at a branching point $(x \in \{A, B, C\},\ y \in \{A, B, C\})$.

3.3. Commands and their execution

A *command* is a signal sequence composed of two signals. There are six commands consisting of signals A, B and C as shown in Table 1. These commands are used for extending or branching a wire. (Commands containing signal D will be explained later).

We assume that, at even time step, two signals of a command are in one cell of a wire. For example, the command sequence on the wire shown in Fig. 9 is AC, BB, AB, BC.

A *head* is an end cell of a wire to which signals flow, and a *tail* is an end cell from which signals flow. Fig. 10 shows a wire with two heads and a tail.

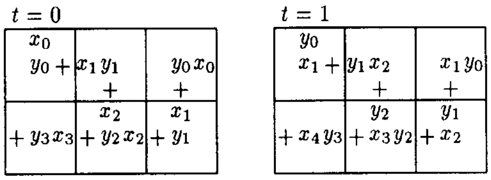

Fig. 8. Signal transmission on a wire with a bifurcation ($x_i, y_i \in \{A, B, C\}$).

Tabel 1
Six commands composed of A, B, and C

Command		Operation
First signal	Second signal	
A	A	Advance the head forward
A	B	Advance the head leftward
A	C	Advance the head rightward
B	A	Branch the wire in three ways
B	B	Branch the wire in two ways (making leftward branch)
B	C	Branch the wire in two ways (making rightward branch)

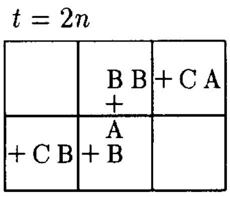

Fig. 9. Command sequence AC, BB, AB, BC travelling on a wire.

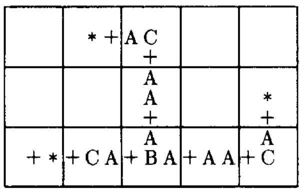

Fig. 10. A wire with two heads (with * in the center part) and a tail (with + in the center part).

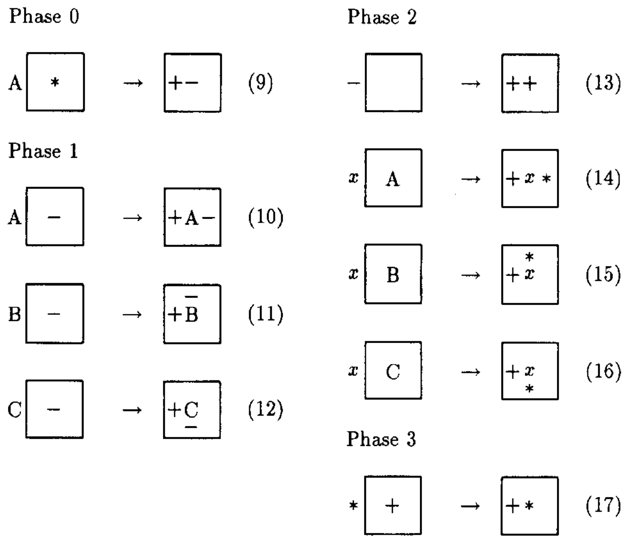

Fig. 11. Rules for advancing a head $(x \in \{A, B, C\})$.

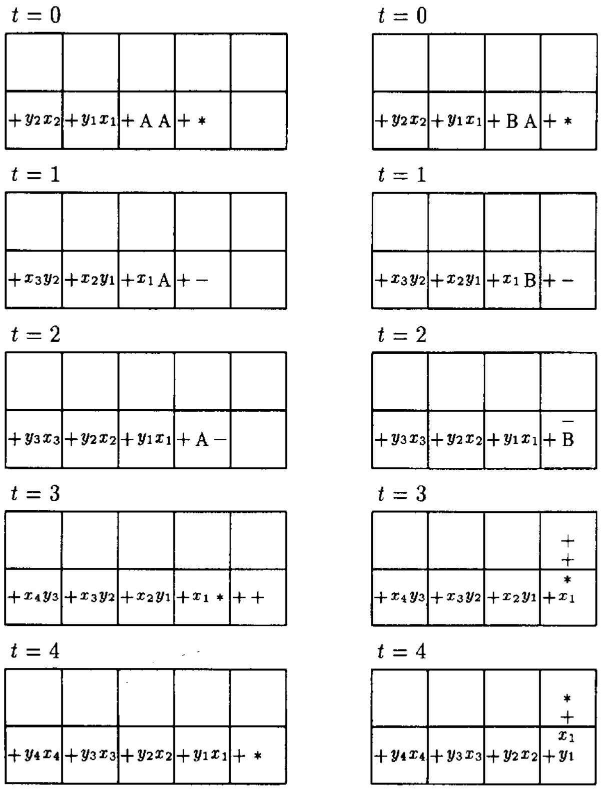

Fig. 12. Execution of advance commands AA (left) and AB (right) $(x_i, y_i \in \{A, B, C\})$.

Six commands shown in Table 1 are "decoded" and "executed" at the head of a wire. Here we explain how the advance commands AA, AB, AC are executed. (Branch commands BA, BB, BC are explained later). Rules (9)–(17) in Fig. 11 are for executing these commands. Since it takes four time steps to execute an advance command, these rules are classified into four groups (from Phase 0 to Phase 3) according to the phase of the execution. An example of this prosess is shown in Fig. 12, where one of the "Phase t" rules in Fig. 11 is applied at time t.

3.4. A Worm

A *Worm* is a simple wire with open ends, thus has a head and a tail. It crawls in the reversible cellular space. As explained in the previous subsection, commands in Table 1 are decoded and executed at the head of a Worm. On the other hand, at the tail cell, the shape of the Worm is "encoded" into an advance command. That is, if the tail of the Worm is straight (or left-turning, right-turning, respectively) in its form, the command AA (AB, AC) is generated. The tail then retracts by one cell. These operations are done by rules (18)–(29) in Fig. 13. Note that in a reversible cellular space, simple retraction (or erasure) of a tail without leaving the shape information is impossible. Therefore, retraction must accompany some

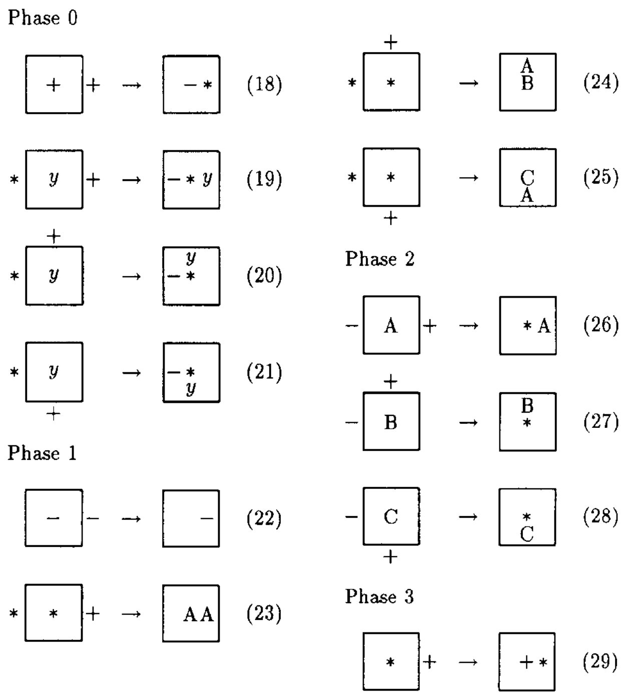

Fig. 13. Rules for retracting a tail ($y \in \{A, B, C\}$).

encoding process. Fig. 14 shows an example of encoding and retracting processes. It takes four time steps to encode and retract a tail by one cell. The command sequence generated by the encoding process may be regarded as a "gene" of the Worm.

It is easy to see that the length (i.e., the number of cells) of a Worm does not vary if it contains only advance commands and never touches itself. Hence it is also easy to see that the same configuration appears infinitely many times if we identify translated configurations as the same ones. Thus, we can classify Worms into two categories: "cycling" and "travelling" ones. A Worm is called *cycling* iff its configuration appears at the same position after some time steps. A Worm is called *travelling* iff its translated configuration appears after some time steps, thus it moves away in some direction. Fig. 15 show examples of cycling and travelling Worms.

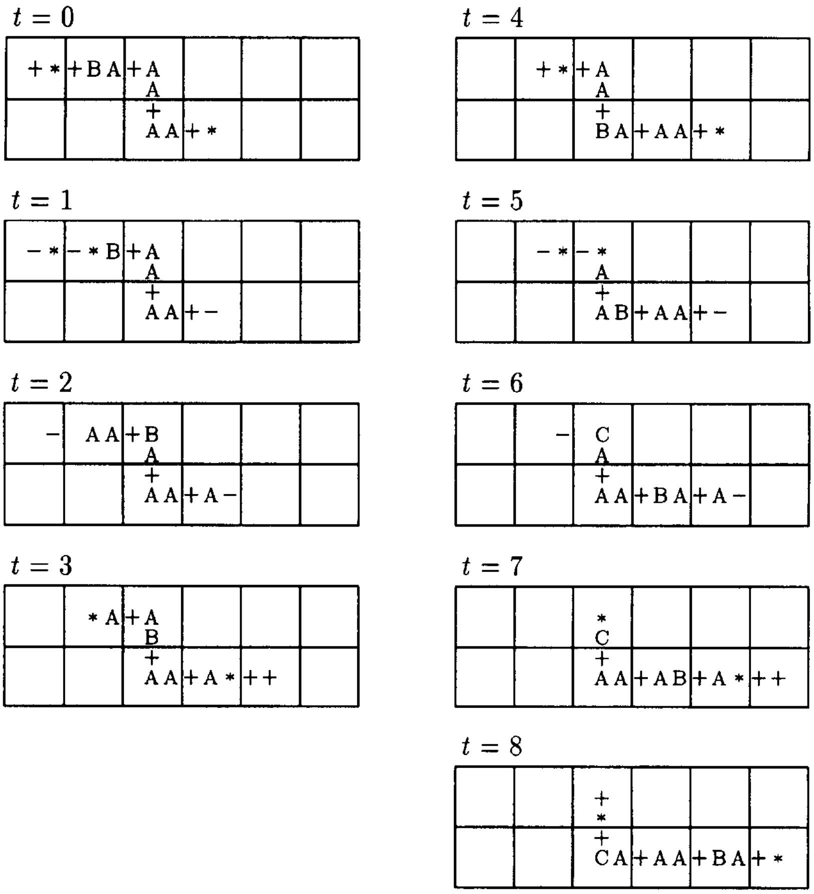

Fig. 14. Retracting process of a tail, where encoding the shape of a tail into advance commands (here, AA and AC are generated) is also carried out.

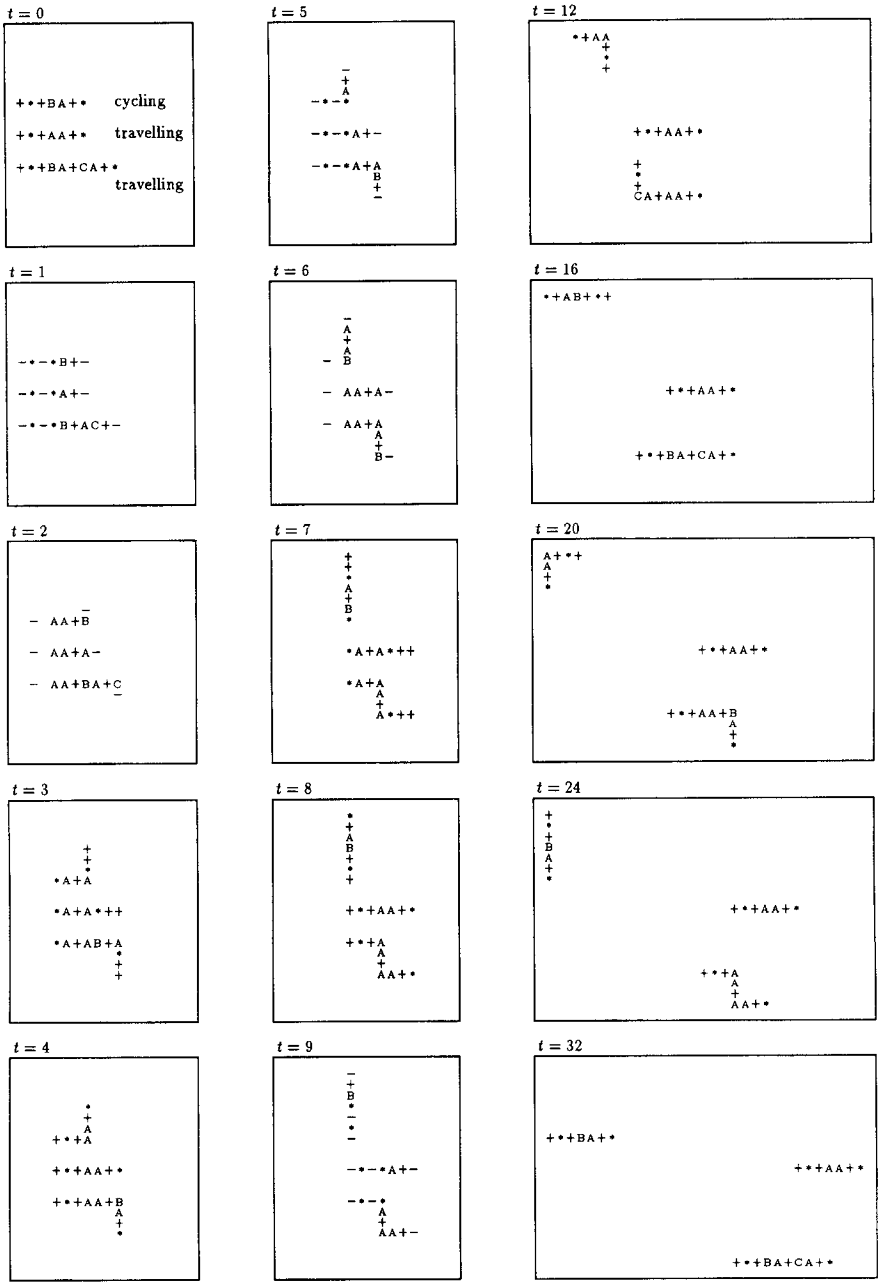

Fig. 15. Behavior of one cycling and two travelling Worms.

3.5. Self-reproduction of a Worm

We now explain branching and splitting processes of a Worm, and then show that a travelling Worm can self-reproduce indefinitely in the two-dimensional plane by giving a branch command. Rules (30)–(37) in Fig. 16 are for executing branch commands BA, BB, BC. A branching process initiated by a command BA is shown in Fig. 17. In the case of a command BB or BC, a Worm branches in two-ways.

At the branching point, a command sequence (or gene) is copied by rules (5)–(8). Hence, these branches have the same shape, though they grow in different directions.

When a tail reaches a branching point, splitting rules (38)–(52) in Fig. 18 are applied.

Fig. 19 shows an example of splitting process caused by a command BA that finally produces three daughter Worms. These Worms also have branch commands, but they

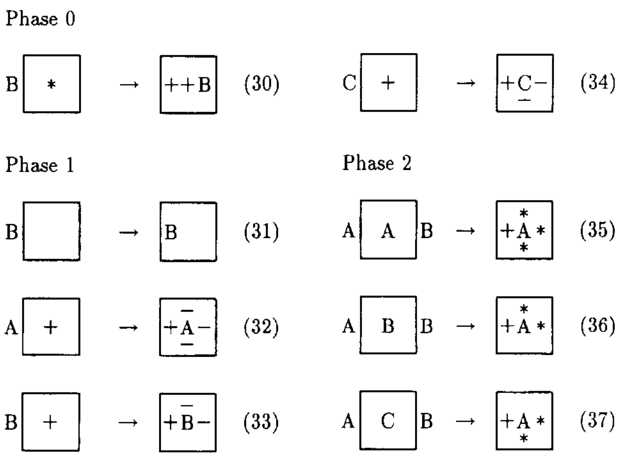

Fig. 16. Rules for branching a head.

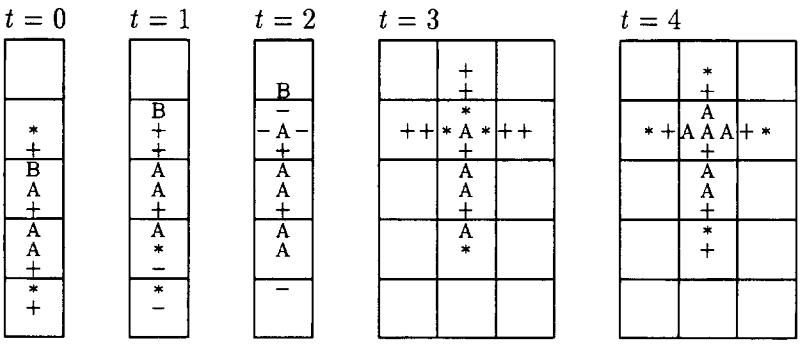

Fig. 17. Execution of a branch command BA.

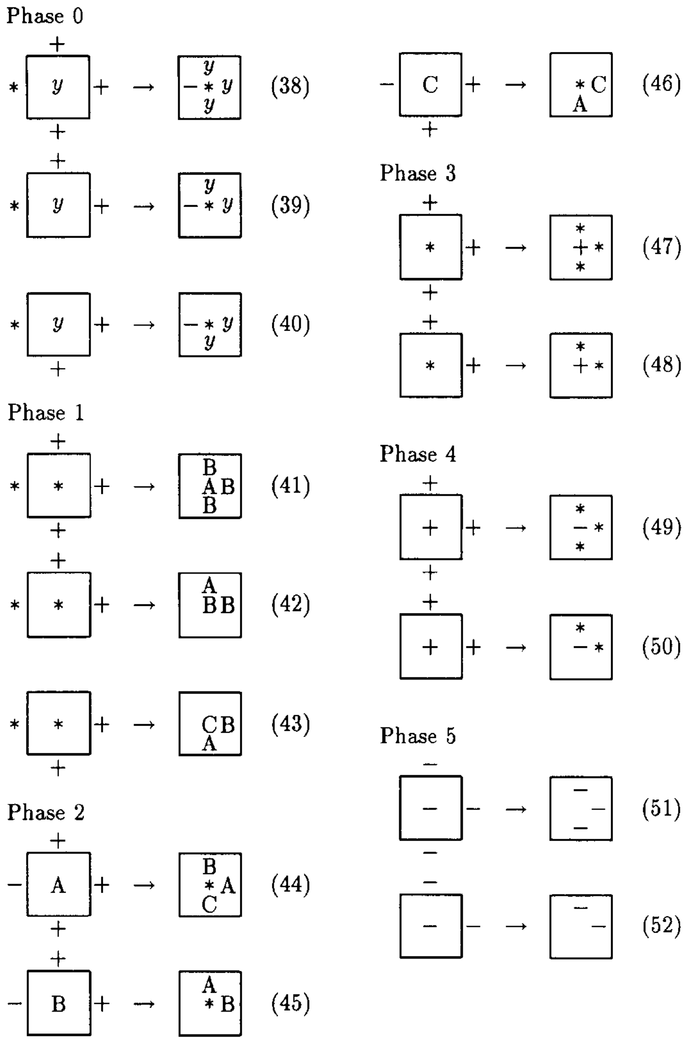

Fig. 18. Rules for splitting a tail ($y \in \{A, B, C\}$).

differ: the center daughter has a three-way branch command BA, while the left and right daughter have a leftward and rightward branch commands BB, BC, respectively, by the rule (44).

Fig. 20 shows a splitting process caused by BB that produces two daughters. In this case, the left daughter has no branch command, while the center one has the leftward branch command BB by rule (45). As we shall see below, overcrowding of Worms can be avoided by this mechanism.

By using branching and splitting mechanism, a travelling Worm with a branch command can self-reproduce indefinitely provided that it does not touch itself in the branching process. Figs. 21 and 22 show self-reproducing processes of Worms.

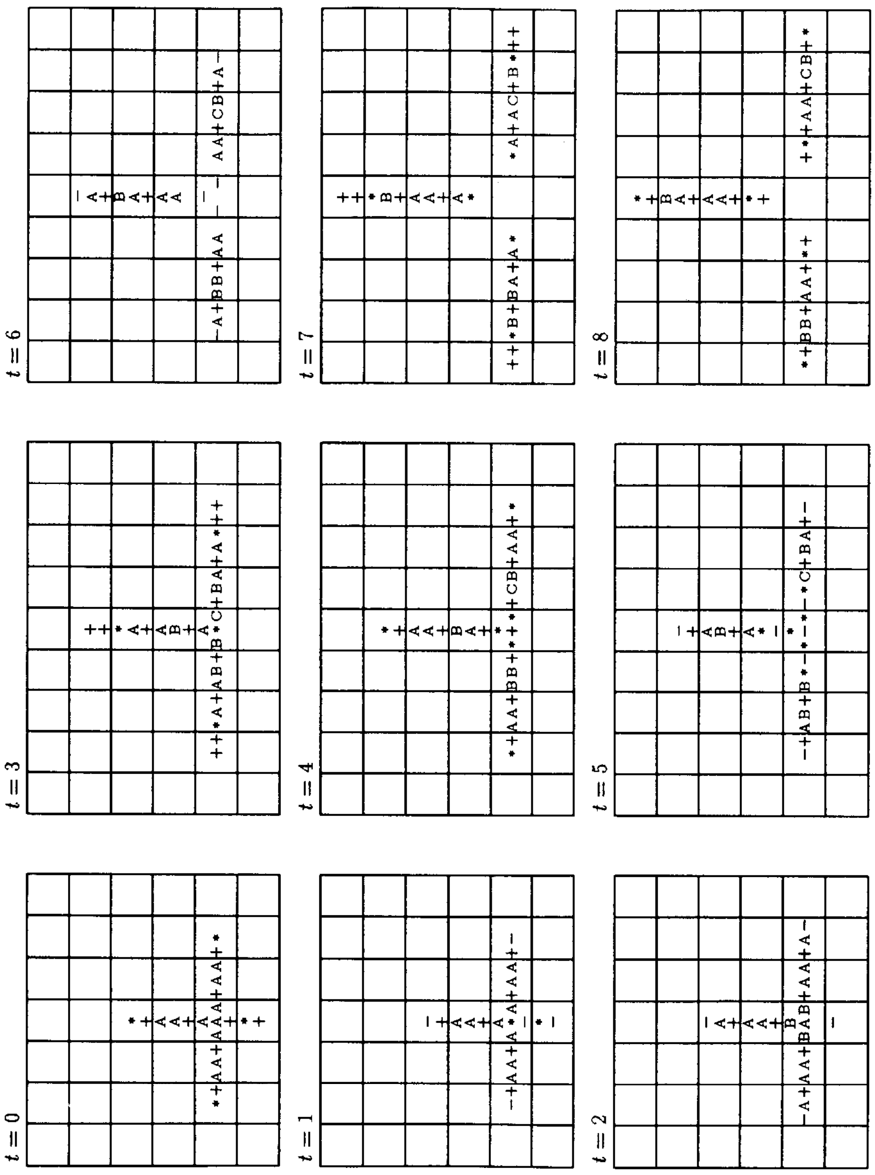

Fig. 19. A splitting process that produces three daughter Worms.

It is easy to see that a travelling Worm (with no branch command) of length n repeats its (translated) configuration every $8(n-2)$ steps. Assume at time 0 there is a configuration of a travelling Worm having only one branch command BA. It also takes $8(n-2)$ steps to become a configuration containing three daughter Worms identical

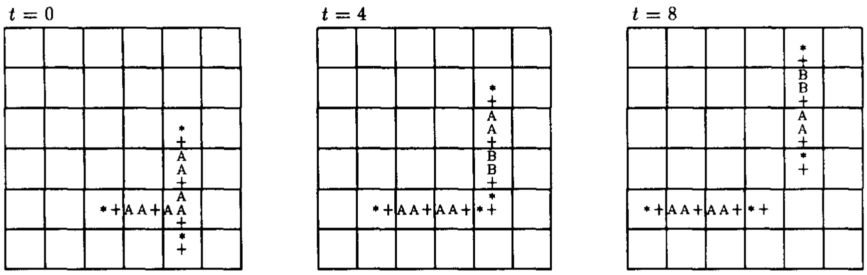

Fig. 20. A splitting process that produces two daughter Worms.

to the initial one except that they may have different branch command and may be rotated. The center daughter further produces three granddaughters. Each of the left and right daughters produces two, and so on. Fig. 23 shows a family of Worms and their positions.

Let $A(k), B(k), C(k)$ and $Z(k)$ be the total number of Worms having a branch command BA, BB, BC, and no branch command, respectively, at time $8(n-2)k$. Then the following equations hold for $k = 0, 1, 2, \ldots$:

$$A(k) = 1$$

$$B(0) = 0, \quad B(k+1) = A(k) + B(k)$$

$$C(0) = 0, \quad C(k+1) = A(k) + C(k)$$

$$Z(0) = 0, \quad Z(k+1) = Z(k) + B(k) + C(k)$$

By solving these equations we can get

$$N(k) \overset{\text{def}}{=} A(k) + B(k) + C(k) + Z(k) = k^2 + k + 1,$$

the total number of Worms at the kth generation.

3.6. Self-reproduction of a loop

A *Loop* is a simple closed wire, thus has neither a head nor a tail as shown in Fig. 24.

If a Loop contains only advance or branch commands, they simply rotate in the Loop and self-reproduction does not occur. In order to make a Loop self-reproduce, commands in Table 2 are used.

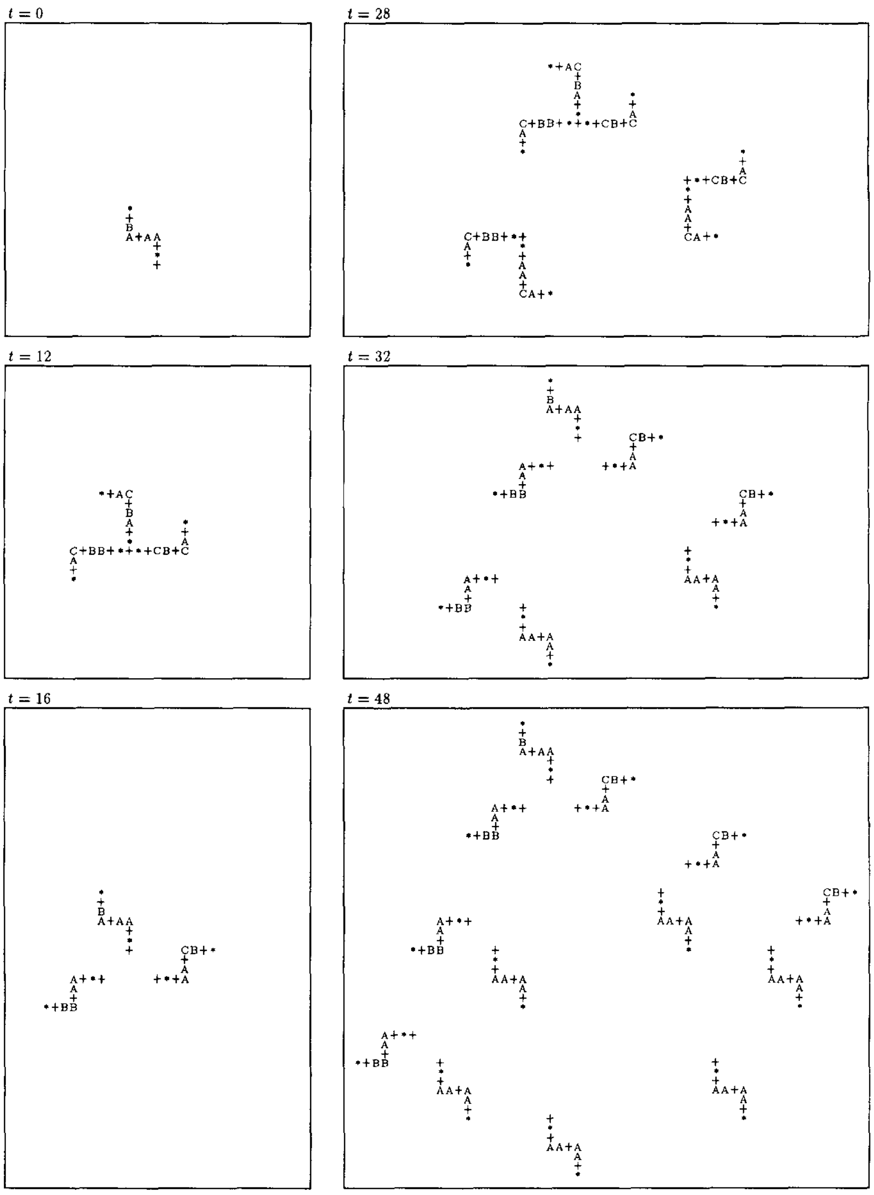

Fig. 21. Self-reproducing process of a Worm (1).

A command DB is transmitted by rules (53)–(58) in Fig. 25. On a straight part of a wire, it propagates in the same way as other commands in Table 1 (rules (53)–(56)). But, at the corner of a wire, it starts to make an "arm" (rules (57) and (58)). An *arm* is a kind of branch to construct an offspring of a Loop. Rules (59)–(67) in Fig. 26 are for creating an arm.

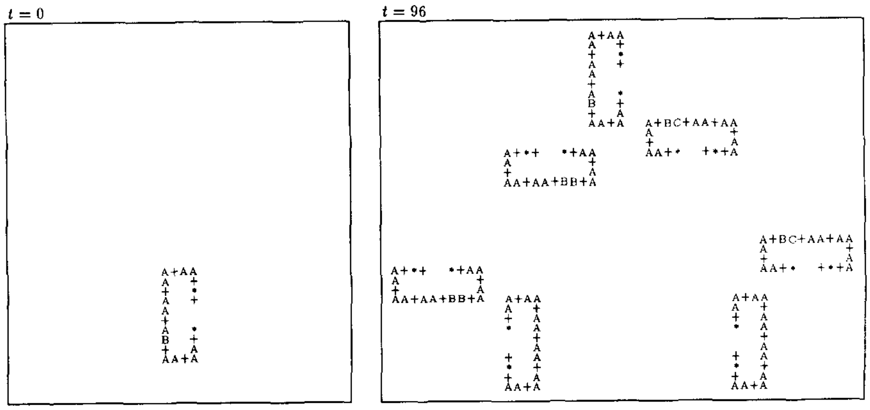

Fig. 22. Self-reproducing process of a Worm (2).

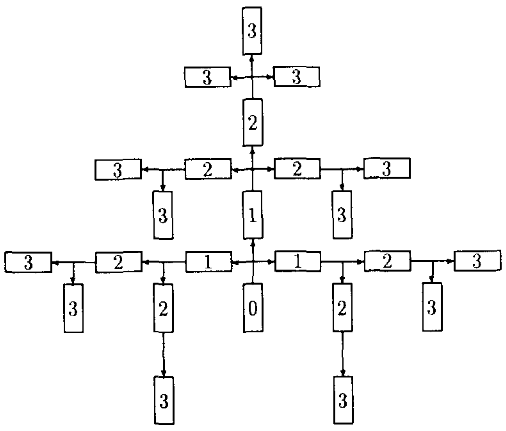

Fig. 23. Lineage of Worms. (The number represents the generation of a Worm.)

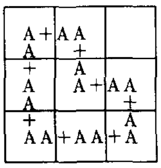

Fig. 24. An example of a Loop.

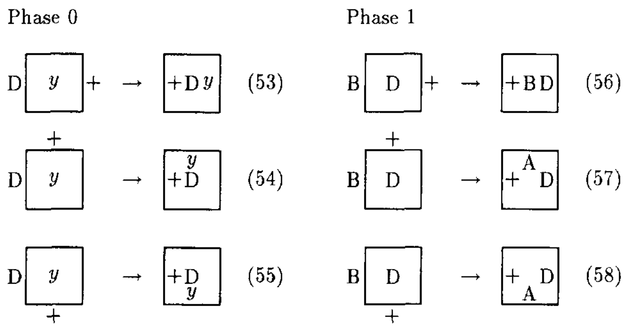

Fig. 25. Rules for transmitting a command DB ($y \in \{A, B, C\}$).

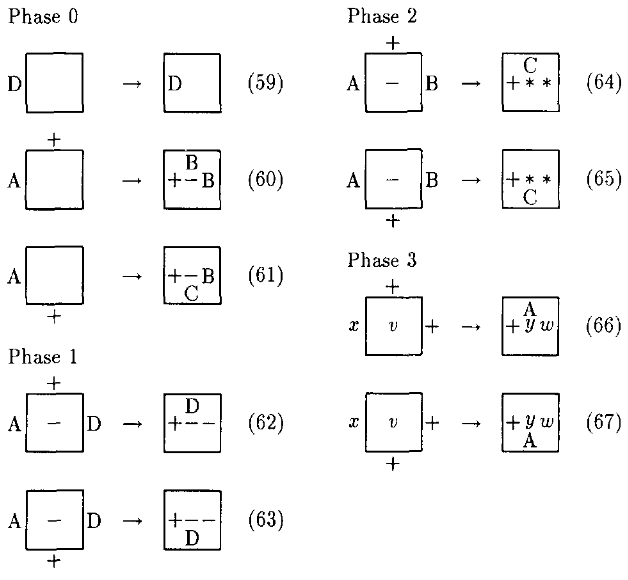

Fig. 26. Rules for creating an arm ($(v,w) \in \{(*, A), (-, B), (+, C)\}$, $(x,y) \in \{(A, *), (B, -), (C, +)\}$).

Table 2
Commands DB and DC

Command		Operation
First signal	Second signal	
D	B	Create an arm
D	C	Encode the shape of a Loop

The root of an arm is a special kind of branching point. Signals reaching this cell are transmitted only to the arm, and signals A's are put into the mother Loop (rules (66) and (67)). Note that at the center part of this branching cell, the states $*, -, +$ are used to represent the signals A, B, C, respectively. Fig. 27 shows a process of transmitting a command DB and creating an arm.

After creating an arm, the form of the mother Loop is encoded into a sequence of advance commands. This process is controlled by a special command DC, which is generated when the arm is created (at time $t = 6$ and 7 in Fig. 27). Encoding is

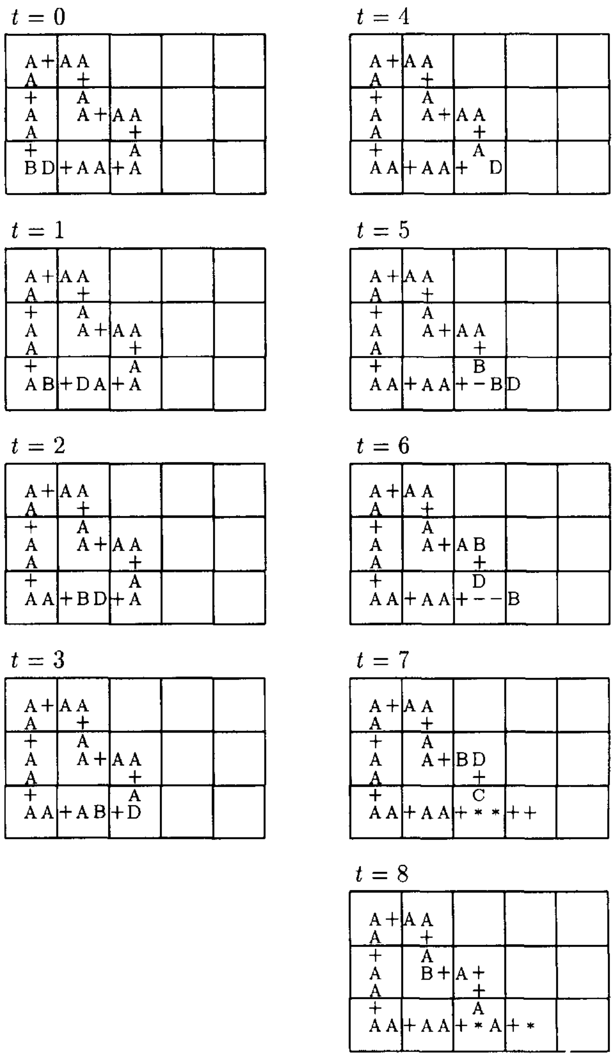

Fig. 27. Processes of transmitting a command DB ($0 \leqslant t \leqslant 4$) and creating an arm ($4 \leqslant t \leqslant 8$).

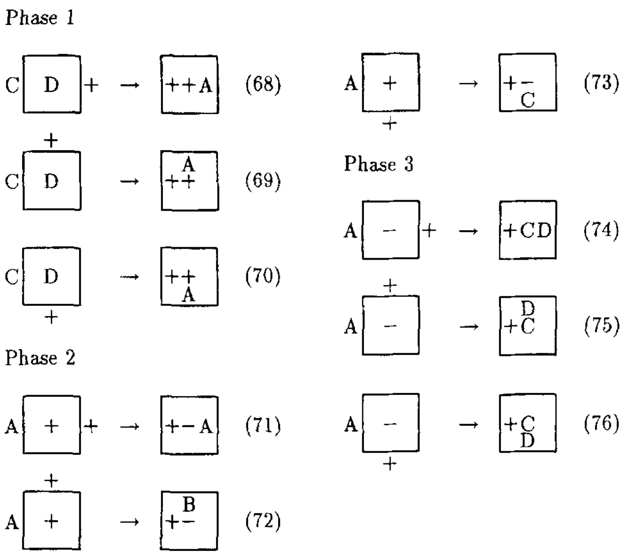

Fig. 28. Rules for encoding the shape of a Loop by a command DC. (Note that rules (53)–(55) are used in Phase 0.)

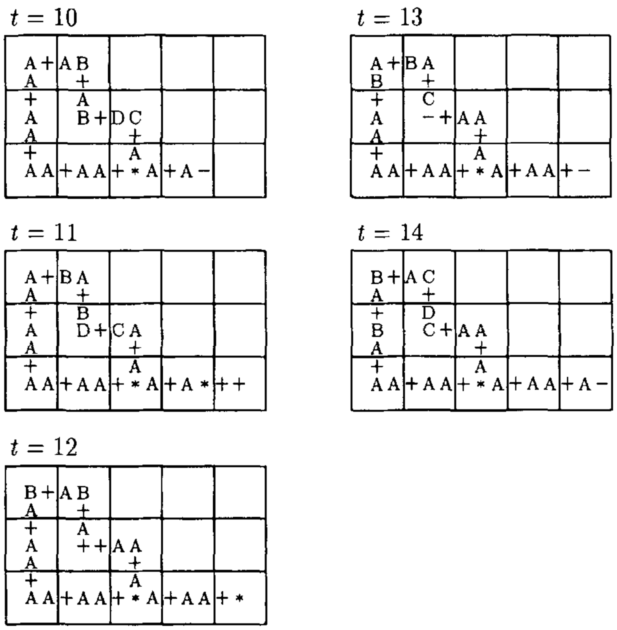

Fig. 29. Encoding the shape of a Loop into an advance command by a command DC. (Advance command AC is generated at $t = 14$.)

performed by rules (68)–(76) in Fig. 28. Generated sequence of advance commands travel through the Loop and then sent to the arm by rules (66) and (67). This command sequence is the gene of the mother Loop. Fig. 29 shows an example of an encoding process.

If the command DC reaches the branching point encoding process terminates. Then the arm is cut off by rules (77)–(84) in Fig. 30. This process is shown in Fig. 31.

The arm cut off from the mother Loop acts just like a Worm. But its head will eventually meet its tail to form a Loop identical to its mother. Rules (85)–(92) in Fig. 32 are for making a daughter Loop, and Fig. 33 shows its process.

An example of an entire self-reproducing process of a Loop is shown in Fig. 34. By putting a command DB at an appropriate position, *every* Loop having only AA commands in the other cells can self-reproduce in this way.

3.7. Reversibility of SR_8

All the rules of SR_8 are listed in the Appendix. Parenthesized number attached to each rule corresponds to the rule number in this section. There are 765 rules in total including rotated ones. Since the rules in the Appendix are sorted on the right-hand

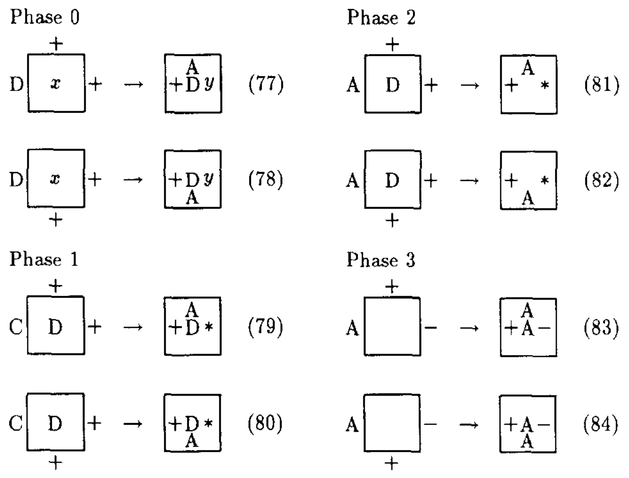

Fig. 30. Rules for cutting off an arm $((x, y) \in \{(*, \mathrm{A}), (-, \mathrm{B}), (+, \mathrm{C})\})$.

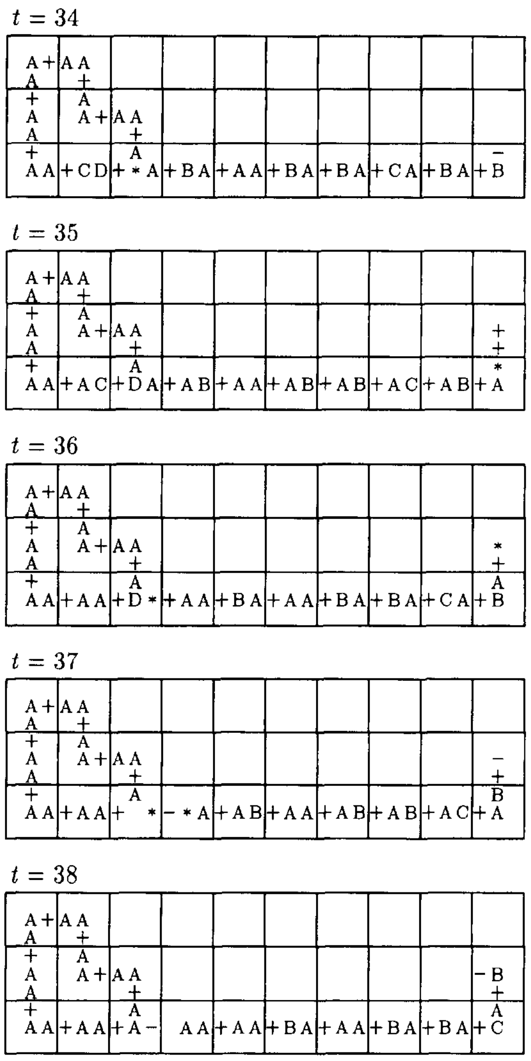

Fig. 31. A process of cutting off an arm.

sides, we can easily verify (but of course tedious!) that there is no pair of distinct rules having the same right-hand side (we also tested it by a computer). Hence, by Propositions 2.1 and 2.3, we can define the local function of SR_8 totally so that it is reversible.

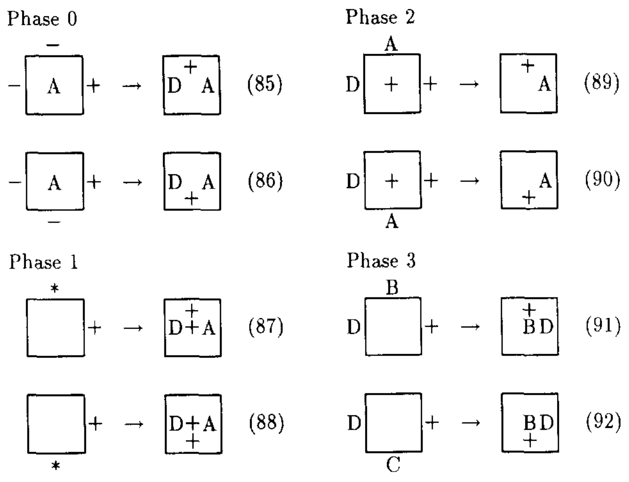

Fig. 32. Rules for making a daughter Loop.

4. Concluding remarks

In this paper, we gave a reversible PCA SR_8, and showed that Worms and Loops can self-reproduce in the reversible cellular space. Since PCA is a subclass of CA (Proposition 2.2), we can also conclude that self-reproduction is possible in a (usual) reversible CA.

In SR_8, conversion between an object itself and its description (gene) can be performed in a very simple manner in both directions. This makes self-reproducing mechanism simple.

The following problems are left for the future study.

1. To design a simpler RPCA (i.e., to reduce the number of states) that supports self-reproduction.
2. To design an RPCA that supports both computation- and construction-universality (i.e., self-reproduction in von Neumann's sense).

Appendix. The set of rules of SR_8

Fig. 35 shows the complete list of 765 rules of SR_8. Rules are sorted on their right-hand sides, and the parenthesized number corresponds to the rule number shown in Section 3.

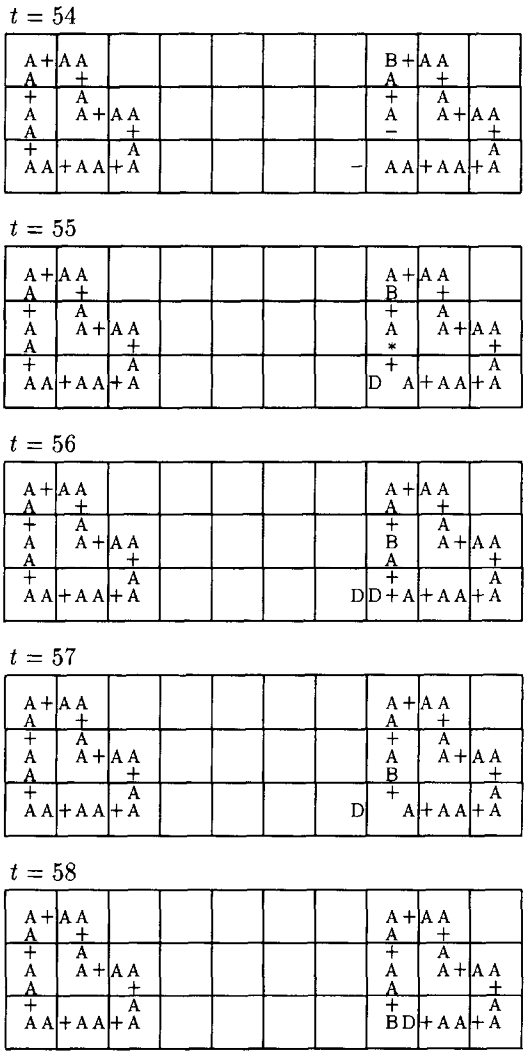

Fig. 33. A process of making a daughter Loop.

$t = 0$

```
A+AA+AA+AA
A        +
+        A
A  AA+A  A
A  +  A  +
+  A  +  A
A  A  AA+A
A  +
+  A
A  A+AA+AA
A        +
+        A
BD+AA+AA+A
```

$t = 50$

```
A+AB+A++AA
A        +
+        A
B  AA+A  A
A  +  A  +
+  A  +  A
B  A  AA+A
A  +
+  A
C  A+AA+AA
A        +
+        A
CA+AA+CA+*A+BA+BA+AA+AA+AA+AA+AA+AA+AA+AA+*
```

$t = 100$

```
A+AA+AA+AA
A        +
+        A
A  AA+A  A
A  +  A  +
+  A  +  A                                                  -
A  A  AA+A                                                  A
A  +                                                        +
+  A                                                        A
A  A+AA+AA                                                  C+AC+AB
A        +                                                        +
+        A                                                        A
AA+AA+AA+A         - AA+AA+AA+AA+AA+AA+BA+AA+AA+AA+BA+AA+AA+BA+AA+B
```

$t = 152$

```
A+AA+AA+AA                                              A+AA+AA+AA
A        +                                              A        +
+        A                                              +        A
A  AA+A  A                                              A  AA+A  A
A  +  A  +                                              A  +  A  +
+  A  +  A                                              +  A  +  A
A  A  AA+A                                              A  A  AA+A
A  +                                                    A  +
+  A                                                    +  A
A  A+AA+AA                                              A  A+AA+AA
A        +                                              A        +
+        A                                              +        A
AA+AA+AA+A                                              BD+AA+AA+A
```

Fig. 34. Self-reproducing process of a Loop (2).

```
[#,#,#,#,#] → [#,#,#,#,#] ( 1)   [B,#,-,+,#] → [*,#,#,B,#] (27)   [C,*,+,#,#] → [*,-,C,#,#] (20)
[-,#,#,#,-] → [#,#,#,#,-] (22)   [B,#,#,+,*] → [*,#,#,B,-] (21)   [C,*,+,+,#] → [*,-,C,C,#] (39)
[#,#,#,#,B] → [#,#,#,#,B] (31)   [C,#,#,+,-] → [*,#,#,C,#] (28)   [C,*,+,+,+] → [*,-,C,C,C] (38)
[#,#,#,#,D] → [#,#,#,#,D] (59)   [C,#,#,+,*] → [*,#,#,C,-] (21)   [A,+,#,-,#] → [*,A,#,#,#] (26)
[+,#,D,A,+] → [#,#,#,+,A] (89)   [C,-,#,+,+] → [*,#,#,C,A] (46)   [A,+,#,#,*] → [*,A,#,#,-] (20)
[-,#,#,-,#] → [#,#,#,-,#] (22)   [-,#,B,+,A] → [*,#,*,C,+] (65)   [C,+,-,#,+] → [*,A,#,#,C] (46)
[-,#,#,-,-] → [#,#,#,-,-] (52)   [+,#,*,#,#] → [*,#,+,#,#] (17)   [*,+,#,A,+] → [*,A,#,+,A] (66)
[+,D,#,+,A] → [#,#,#,A,+] (90)   [*,#,A,+,+] → [*,#,+,A,A] (66)   [A,+,#,*,#] → [*,A,#,-,#] (19)
[#,#,#,B,#] → [#,#,#,B,#] (31)   [-,#,A,+,+] → [*,#,+,A,B] (66)   [A,+,#,*,+] → [*,A,#,-,A] (39)
[#,#,#,D,#] → [#,#,#,D,#] (59)   [+,#,A,+,+] → [*,#,+,A,C] (66)   [*,+,A,#,+] → [*,A,+,#,A] (67)
[D,#,+,+,A] → [#,#,*,A,+] (82)   [-,#,A,+,B] → [*,#,+,C,*] (64)   [-,+,A,#,+] → [*,A,+,#,B] (67)
[+,D,A,+,#] → [#,#,+,A,#] (89)   [A,#,*,#,+] → [*,#,-,#,A] (19)   [+,+,A,#,+] → [*,A,+,#,C] (67)
[D,#,A,+,+] → [#,#,+,A,*] (81)   [B,#,*,#,+] → [*,#,-,#,B] (19)   [A,+,*,#,#] → [*,A,-,#,#] (21)
[D,#,B,+,#] → [#,#,+,A,D] (57)   [C,#,*,#,+] → [*,#,-,#,C] (19)   [A,+,*,#,+] → [*,A,-,#,A] (40)
[-,#,-,#,#] → [#,#,-,#,#] (22)   [A,#,*,+,#] → [*,#,-,A,#] (20)   [A,+,*,+,+] → [*,A,-,A,A] (38)
[-,#,-,-,#] → [#,#,-,-,#] (52)   [A,#,*,+,+] → [*,#,-,A,A] (39)   [*,+,+,#,A] → [*,A,A,#,+] (66)
[-,#,-,-,-] → [#,#,-,-,-] (51)   [B,#,*,+,#] → [*,#,-,B,#] (20)   [A,+,+,#,*] → [*,A,A,#,-] (39)
[+,#,+,A,D] → [#,#,A,+,#] (90)   [B,#,*,+,+] → [*,#,-,B,B] (39)   [*,+,+,A,#] → [*,A,A,+,#] (67)
[A,#,+,-,-] → [#,#,A,+,D] (86)   [C,#,*,+,#] → [*,#,-,C,#] (20)   [A,+,+,*,#] → [*,A,A,-,#] (40)
[#,#,B,#,#] → [#,#,B,#,#] (31)   [C,#,*,+,+] → [*,#,-,C,C] (39)   [A,+,+,*,+] → [*,A,A,-,A] (38)
[#,#,D,#,#] → [#,#,D,#,#] (59)   [A,#,+,#,-] → [*,#,A,#,#] (26)   [A,+,+,+,*] → [*,A,A,A,-] (38)
[A,#,-,-,+] → [#,#,D,+,A] (85)   [A,#,+,#,*] → [*,#,A,#,-] (19)   [B,+,+,#,-] → [*,A,B,#,#] (45)
[D,#,#,+,B] → [#,#,D,A,+] (58)   [A,#,+,*,#] → [*,#,A,-,#] (21)   [-,+,+,#,A] → [*,A,B,#,+] (66)
[D,+,#,A,+] → [#,*,#,+,A] (81)   [*,#,+,+,A] → [*,#,A,A,+] (67)   [+,+,+,#,A] → [*,A,C,#,+] (66)
[D,+,+,A,#] → [#,*,A,+,#] (82)   [A,#,+,+,*] → [*,#,A,A,-] (40)   [A,+,+,-,+] → [*,A,C,#,B] (44)
[+,A,D,#,+] → [#,+,#,#,A] (90)   [B,-,+,+,#] → [*,#,A,B,#] (45)   [B,+,#,#,-] → [*,B,#,#,#] (27)
[D,A,#,+,+] → [#,+,#,*,A] (82)   [B,-,+,#,#] → [*,#,B,#,#] (27)   [B,+,#,#,*] → [*,B,#,#,-] (20)
[D,B,#,#,+] → [#,+,#,D,A] (58)   [B,#,+,#,*] → [*,#,B,#,-] (19)   [B,+,#,-,+] → [*,B,#,#,A] (45)
[+,A,+,#,D] → [#,+,A,#,#] (89)   [B,#,+,*,#] → [*,#,B,-,#] (21)   [-,+,#,A,+] → [*,B,#,+,A] (66)
[A,-,+,#,-] → [#,+,A,#,D] (85)   [-,#,+,+,A] → [*,#,B,A,+] (67)   [B,+,#,*,#] → [*,B,#,-,#] (19)
[D,A,+,+,#] → [#,+,A,*,#] (81)   [A,-,+,+,+] → [*,#,B,A,C] (44)   [B,+,#,*,+] → [*,B,#,-,B] (39)
[D,B,+,#,#] → [#,+,A,D,#] (57)   [B,#,+,+,*] → [*,#,B,B,-] (40)   [B,+,*,#,#] → [*,B,-,#,#] (21)
[A,-,-,#,+] → [#,+,D,#,A] (86)   [C,#,+,-,#] → [*,#,C,#,#] (28)   [B,+,*,#,+] → [*,B,-,#,B] (40)
[-,-,#,#,#] → [#,-,#,#,#] (22)   [C,#,+,#,*] → [*,#,C,#,-] (19)   [B,+,*,+,+] → [*,B,-,B,B] (38)
[-,-,#,#,-] → [#,-,#,#,-] (52)   [C,#,+,*,#] → [*,#,C,-,#] (21)   [-,+,+,A,#] → [*,B,A,+,#] (67)
[-,-,#,-,-] → [#,-,#,-,-] (51)   [C,#,+,+,-] → [*,#,C,A,#] (46)   [A,+,+,+,-] → [*,B,A,C,#] (44)
[-,-,-,#,#] → [#,-,-,#,#] (52)   [+,#,+,+,A] → [*,#,C,A,+] (67)   [B,+,+,#,*] → [*,B,B,#,-] (39)
[-,-,-,#,-] → [#,-,-,#,-] (51)   [C,#,+,+,*] → [*,#,C,C,-] (40)   [B,+,+,*,#] → [*,B,B,-,#] (40)
[-,-,-,-,#] → [#,-,-,-,#] (51)   [-,B,#,A,+] → [*,*,#,+,C] (64)   [B,+,+,*,+] → [*,B,B,-,B] (38)
[+,+,#,D,A] → [#,A,#,#,+] (89)   [-,B,+,A,#] → [*,*,C,+,#] (65)   [B,+,+,+,*] → [*,B,B,B,-] (38)
[A,+,#,-,-] → [#,A,#,D,+] (85)   [+,*,#,#,#] → [*,+,#,#,#] (17)   [C,+,-,#,#] → [*,C,#,#,#] (28)
[D,+,+,#,A] → [#,A,*,#,+] (81)   [-,A,#,B,+] → [*,+,#,*,C] (65)   [C,+,#,#,*] → [*,C,#,#,-] (20)
[+,+,A,D,#] → [#,A,+,#,#] (90)   [*,A,#,+,+] → [*,+,#,A,A] (67)   [+,+,#,A,+] → [*,C,#,+,A] (66)
[D,+,A,#,+] → [#,A,+,#,*] (82)   [-,A,#,+,+] → [*,+,#,B,A] (67)   [C,+,#,*,#] → [*,C,#,-,#] (19)
[D,+,B,#,#] → [#,A,+,#,D] (58)   [+,A,#,+,+] → [*,+,#,C,A] (67)   [C,+,#,*,+] → [*,C,#,-,C] (39)
[A,+,-,-,#] → [#,A,+,D,#] (86)   [*,A,+,+,#] → [*,+,A,A,#] (66)   [A,+,-,+,+] → [*,C,#,B,A] (44)
[D,+,#,#,B] → [#,A,D,#,+] (57)   [-,A,+,+,#] → [*,+,A,B,#] (66)   [-,+,B,#,A] → [*,C,*,#,+] (64)
[#,B,#,#,#] → [#,B,#,#,#] (31)   [+,A,+,+,#] → [*,+,A,C,#] (66)   [-,+,A,#,B] → [*,C,+,#,*] (65)
[#,D,#,#,#] → [#,D,#,#,#] (59)   [-,A,+,B,#] → [*,+,C,*,#] (64)   [C,+,*,#,#] → [*,C,-,#,#] (21)
[D,#,#,B,+] → [#,D,#,+,A] (57)   [A,*,#,#,+] → [*,-,#,#,A] (21)   [C,+,*,#,+] → [*,C,-,#,C] (40)
[A,-,#,+,-] → [#,D,#,A,+] (86)   [B,*,#,#,+] → [*,-,#,#,B] (21)   [C,+,*,+,+] → [*,C,-,C,C] (38)
[A,-,-,+,#] → [#,D,+,A,#] (85)   [C,*,#,#,+] → [*,-,#,#,C] (21)   [C,+,+,-,#] → [*,C,A,#,#] (46)
[D,#,+,B,#] → [#,D,A,+,#] (58)   [A,*,#,+,#] → [*,-,#,A,#] (19)   [+,+,+,A,#] → [*,C,A,+,#] (67)
[+,#,#,#,*] → [*,#,#,#,+] (17)   [A,*,#,+,+] → [*,-,#,A,A] (40)   [C,+,+,#,*] → [*,C,C,#,-] (39)
[A,#,-,#,+] → [*,#,#,#,A] (26)   [B,*,#,+,#] → [*,-,#,B,#] (19)   [C,+,+,*,#] → [*,C,C,-,#] (40)
[B,#,#,-,+] → [*,#,#,#,B] (27)   [B,*,#,+,+] → [*,-,#,B,B] (40)   [C,+,+,*,+] → [*,C,C,-,C] (38)
[C,-,#,#,+] → [*,#,#,#,C] (28)   [C,*,#,+,#] → [*,-,#,C,#] (19)   [C,+,+,+,*] → [*,C,C,C,-] (38)
[+,#,#,*,#] → [*,#,#,+,#] (17)   [C,*,#,+,+] → [*,-,#,C,C] (40)   [*,#,#,#,+] → [+,#,#,#,*] (29)
[A,#,#,*,+] → [*,#,#,-,A] (20)   [A,*,+,#,#] → [*,-,A,#,#] (20)   [#,#,#,#,-] → [+,#,#,#,+] (13)
[B,#,#,*,+] → [*,#,#,-,B] (20)   [A,*,+,+,#] → [*,-,A,A,#] (39)   [*,#,#,+,#] → [+,#,#,*,#] (29)
[C,#,#,*,+] → [*,#,#,-,C] (20)   [A,*,+,+,+] → [*,-,A,A,A] (38)   [*,#,#,+,+] → [+,#,#,*,*] (48)
[A,-,#,+,#] → [*,#,#,A,#] (26)   [B,*,+,#,#] → [*,-,B,#,#] (20)   [#,#,#,-,#] → [+,#,#,+,#] (13)
[A,#,#,+,*] → [*,#,#,A,-] (21)   [B,*,+,+,#] → [*,-,B,B,#] (39)   [D,#,#,C,+] → [+,#,#,+,A] (69)
[B,#,-,+,+] → [*,#,#,A,B] (45)   [B,*,+,+,+] → [*,-,B,B,B] (38)   [D,#,#,+,C] → [+,#,#,A,+] (70)
```

Fig. 35. Complete list of 765 rules of SR_8.

```
[*,#,+,#,#] → [+,#,*,#,#] (29)   [+,#,+,#,#] → [-,#,*,#,#] (18)   [A,#,#,A,+] → [A,#,#,+,A] ( 3)
[*,#,+,+,#] → [+,#,*,*,#] (48)   [+,#,+,+,#] → [-,#,*,*,#] (50)   [B,#,#,A,+] → [A,#,#,+,B] ( 3)
[*,#,+,+,+] → [+,#,*,*,*] (47)   [+,#,+,+,+] → [-,#,*,*,*] (49)   [C,#,#,A,+] → [A,#,#,+,C] ( 3)
[#,#,-,#,#] → [+,#,+,#,#] (13)   [*,#,A,#,#] → [-,#,+,#,#] ( 9)   [*,*,#,+,#] → [A,#,#,A,#] (23)
[D,#,C,#,+] → [+,#,+,#,A] (68)   [+,#,A,#,+] → [-,#,+,#,A] (71)   [A,#,#,+,A] → [A,#,#,A,+] ( 4)
[*,#,B,#,#] → [+,#,+,#,B] (30)   [*,#,B,+,+] → [-,#,+,A,A] (66)   [B,#,#,+,A] → [A,#,#,B,+] ( 4)
[D,#,C,+,#] → [+,#,+,A,#] (69)   [-,#,B,+,+] → [-,#,+,A,B] (66)   [C,#,#,+,A] → [A,#,#,C,+] ( 4)
[*,#,C,+,+] → [+,#,+,A,A] (66)   [+,#,B,+,+] → [-,#,+,A,C] (66)   [A,#,#,#,A] → [A,#,*,#,+] (14)
[-,#,C,+,+] → [+,#,+,A,B] (66)   [+,#,A,+,#] → [-,#,+,B,#] (72)   [C,#,B,#,A] → [A,#,*,*,+] (37)
[+,#,C,+,+] → [+,#,+,A,C] (66)   [#,#,A,+,#] → [-,#,+,B,B] (60)   [C,#,#,A,#] → [A,#,*,+,#] (16)
[D,#,+,#,C] → [+,#,A,#,+] (68)   [-,#,A,+,D] → [-,#,+,D,-] (62)   [A,#,A,#,#] → [A,#,+,#,*] (14)
[D,#,+,C,#] → [+,#,A,+,#] (70)   [-,#,D,+,A] → [-,#,-,D,+] (63)   [-,#,A,#,#] → [A,#,+,#,-] (10)
[#,#,+,*,#] → [+,#,A,+,D] (88)   [+,#,+,#,A] → [-,#,A,#,+] (71)   [A,#,A,#,+] → [A,#,+,#,A] ( 2)
[*,#,+,+,C] → [+,#,A,A,+] (67)   [*,#,+,+,B] → [-,#,A,A,+] (67)   [B,#,A,#,+] → [A,#,+,#,B] ( 2)
[*,#,#,#,B] → [+,#,B,#,+] (30)   [-,#,+,+,B] → [-,#,B,A,+] (67)   [C,#,A,#,+] → [A,#,+,#,C] ( 2)
[-,#,+,+,C] → [+,#,B,A,+] (67)   [#,#,#,+,A] → [-,#,B,C,+] (61)   [B,#,A,#,#] → [A,#,+,*,#] (15)
[+,#,+,+,C] → [+,#,C,A,+] (67)   [+,#,+,A,#] → [-,#,C,+,#] (73)   [B,#,A,#,B] → [A,#,+,*,*] (36)
[#,#,#,*,+] → [+,#,D,+,A] (87)   [+,#,+,+,B] → [-,#,C,A,+] (67)   [A,#,A,+,#] → [A,#,+,A,#] ( 3)
[*,+,#,#,#] → [+,*,#,#,#] (29)   [+,+,#,#,#] → [-,*,#,#,#] (18)   [#,#,A,+,-] → [A,#,+,A,-] (83)
[*,+,#,#,+] → [+,*,#,#,*] (48)   [+,+,#,#,+] → [-,*,#,#,*] (50)   [A,#,A,+,+] → [A,#,+,A,A] ( 5)
[*,+,#,+,+] → [+,*,#,*,*] (47)   [+,+,#,+,+] → [-,*,#,*,*] (49)   [B,#,A,+,#] → [A,#,+,B,#] ( 3)
[*,+,+,#,#] → [+,*,*,#,#] (48)   [+,+,+,#,#] → [-,*,*,#,#] (50)   [B,#,A,+,+] → [A,#,+,B,B] ( 5)
[*,+,+,#,+] → [+,*,*,#,*] (47)   [+,+,+,#,+] → [-,*,*,#,*] (49)   [C,#,A,+,#] → [A,#,+,C,#] ( 3)
[*,+,+,+,#] → [+,*,*,*,#] (47)   [+,+,+,+,#] → [-,*,*,*,#] (49)   [C,#,A,+,+] → [A,#,+,C,C] ( 5)
[#,-,#,#,#] → [+,+,#,#,#] (13)   [*,A,#,#,#] → [-,+,#,#,#] ( 9)   [-,#,#,#,A] → [A,#,-,#,+] (10)
[D,C,#,#,+] → [+,+,#,#,A] (70)   [+,A,#,#,+] → [-,+,#,#,C] (73)   [#,#,-,+,A] → [A,#,-,A,+] (84)
[D,C,#,+,#] → [+,+,#,A,#] (68)   [-,A,#,D,+] → [-,+,#,-,D] (63)   [*,#,+,#,*] → [A,#,A,#,#] (23)
[*,C,#,+,+] → [+,+,#,A,A] (67)   [+,A,#,+,#] → [-,+,#,A,#] (71)   [A,#,+,#,A] → [A,#,A,#,+] ( 2)
[*,B,#,#,#] → [+,+,#,B,#] (30)   [*,B,#,+,+] → [-,+,#,A,A] (67)   [A,#,+,A,#] → [A,#,A,+,#] ( 4)
[-,C,#,+,+] → [+,+,#,B,A] (67)   [-,B,#,+,+] → [-,+,#,B,A] (67)   [A,#,+,A,+] → [A,#,A,+,A] ( 7)
[+,C,#,+,+] → [+,+,#,C,A] (67)   [#,A,#,#,+] → [-,+,#,B,C] (61)   [A,#,+,+,A] → [A,#,A,A,+] ( 6)
[D,C,+,#,#] → [+,+,A,#,#] (69)   [+,B,#,+,+] → [-,+,#,C,A] (67)   [B,#,+,#,A] → [A,#,B,#,+] ( 2)
[#,*,+,#,#] → [+,+,A,#,D] (87)   [*,B,+,+,#] → [-,+,A,A,#] (66)   [B,#,+,A,#] → [A,#,B,+,#] ( 4)
[*,C,+,+,#] → [+,+,A,A,#] (66)   [-,B,+,+,#] → [-,+,A,B,#] (66)   [B,#,+,A,+] → [A,#,B,+,B] ( 7)
[-,C,+,+,#] → [+,+,A,B,#] (66)   [+,B,+,+,#] → [-,+,A,C,#] (66)   [B,#,+,+,A] → [A,#,B,B,+] ( 6)
[+,C,+,+,#] → [+,+,A,C,#] (66)   [+,A,+,#,#] → [-,+,B,#,#] (72)   [*,*,+,+,+] → [A,#,B,B,B] (41)
[#,*,#,#,+] → [+,+,D,#,A] (88)   [#,A,+,#,#] → [-,+,B,B,#] (60)   [C,#,+,#,A] → [A,#,C,#,+] ( 2)
[D,+,#,#,C] → [+,A,#,#,+] (69)   [-,A,+,D,#] → [-,+,D,-,#] (62)   [C,#,+,A,#] → [A,#,C,+,#] ( 4)
[D,+,#,C,#] → [+,A,#,+,#] (68)   [-,D,#,A,+] → [-,-,#,+,D] (62)   [C,#,+,A,+] → [A,#,C,+,C] ( 7)
[*,+,#,C,+] → [+,A,#,+,A] (66)   [-,D,+,A,#] → [-,-,D,+,#] (63)   [C,#,+,+,A] → [A,#,C,C,+] ( 6)
[#,+,#,#,*] → [+,A,#,D,+] (87)   [+,+,#,A,#] → [-,A,#,+,#] (71)   [B,#,#,#,A] → [A,*,#,#,+] (15)
[D,+,C,#,#] → [+,A,+,#,#] (70)   [*,+,#,B,+] → [-,A,#,+,A] (66)   [A,#,#,A,#] → [A,*,#,+,#] (14)
[*,+,C,#,+] → [+,A,+,#,A] (67)   [*,+,B,#,+] → [-,A,+,#,A] (67)   [B,B,#,A,#] → [A,*,#,+,*] (36)
[-,+,C,#,+] → [+,A,+,#,B] (67)   [-,+,B,#,+] → [-,A,+,#,B] (67)   [B,#,B,#,A] → [A,*,*,#,+] (36)
[+,+,C,#,+] → [+,A,+,#,C] (67)   [+,+,B,#,+] → [-,A,+,#,C] (67)   [A,#,B,#,A] → [A,*,*,*,+] (35)
[#,+,*,#,#] → [+,A,+,D,#] (88)   [*,+,+,#,B] → [-,A,A,#,+] (66)   [C,B,#,A,#] → [A,*,*,+,#] (37)
[*,+,+,#,C] → [+,A,A,#,+] (66)   [*,+,+,B,#] → [-,A,A,+,#] (67)   [A,B,#,A,#] → [A,*,*,+,*] (35)
[*,+,+,C,#] → [+,A,A,+,#] (67)   [-,+,+,#,B] → [-,A,B,#,+] (66)   [C,#,A,#,#] → [A,*,+,#,#] (16)
[-,+,+,#,C] → [+,A,B,#,+] (66)   [+,+,+,#,B] → [-,A,C,#,+] (66)   [C,#,A,#,B] → [A,*,+,#,*] (37)
[+,+,+,#,C] → [+,A,C,#,+] (66)   [+,+,#,#,A] → [-,B,#,#,+] (72)   [A,#,A,#,B] → [A,*,+,*,*] (35)
[*,#,#,B,#] → [+,B,#,+,#] (30)   [-,+,#,B,+] → [-,B,#,+,A] (66)   [C,A,#,#,#] → [A,+,#,#,*] (16)
[-,+,#,C,+] → [+,B,#,+,A] (66)   [#,#,#,A,+] → [-,B,#,+,B] (60)   [A,A,#,#,+] → [A,+,#,#,A] ( 4)
[-,+,+,C,#] → [+,B,A,+,#] (67)   [-,+,+,B,#] → [-,B,A,+,#] (67)   [B,A,#,#,+] → [A,+,#,#,B] ( 4)
[+,+,#,C,+] → [+,C,#,+,A] (66)   [#,+,#,#,A] → [-,B,B,#,+] (60)   [C,A,#,#,+] → [A,+,#,#,C] ( 4)
[+,+,+,C,#] → [+,C,A,+,#] (67)   [#,#,+,A,#] → [-,B,C,+,#] (61)   [A,A,#,#,#] → [A,+,#,*,#] (14)
[#,#,#,+,*] → [+,D,#,A,+] (88)   [+,+,#,B,+] → [-,C,#,+,A] (66)   [C,A,#,B,#] → [A,+,#,*,*] (37)
[#,#,*,+,#] → [+,D,+,A,#] (87)   [+,+,A,#,#] → [-,C,+,#,#] (73)   [-,A,#,#,#] → [A,+,#,-,#] (10)
[+,#,#,#,+] → [-,#,#,#,*] (18)   [#,+,A,#,#] → [-,C,+,#,B] (61)   [#,A,#,-,+] → [A,+,#,-,A] (84)
[*,#,#,#,A] → [-,#,#,#,+] ( 9)   [+,+,+,B,#] → [-,C,A,+,#] (67)   [A,A,#,+,#] → [A,+,#,A,#] ( 2)
[+,#,#,+,#] → [-,#,#,*,#] (18)   [-,+,A,#,D] → [-,D,+,#,-] (63)   [A,A,#,+,+] → [A,+,#,A,A] ( 6)
[+,#,#,+,+] → [-,#,#,*,*] (50)   [-,+,D,#,A] → [-,D,-,#,+] (62)   [B,A,#,+,#] → [A,+,#,B,#] ( 2)
[*,#,#,A,#] → [-,#,#,+,#] ( 9)   [*,#,*,#,+] → [A,#,#,#,A] (23)   [B,A,#,+,+] → [A,+,#,B,B] ( 6)
[+,#,#,A,+] → [-,#,#,+,B] (72)   [C,#,#,#,A] → [A,#,#,*,+] (16)   [C,A,#,+,#] → [A,+,#,C,#] ( 2)
[+,#,#,+,A] → [-,#,#,C,+] (73)   [B,#,#,A,#] → [A,#,#,+,*] (15)   [C,A,#,+,+] → [A,+,#,C,C] ( 6)
```

Fig. 35. (Continued)

```
[B,A,#,#,#] → [A,+,*,#,#] (15)   [C,+,+,+,A] → [A,C,C,C,+] ( 8)   [B,B,#,#,#] → [B,+,*,#,#] (15)
[B,A,#,B,#] → [A,+,*,*,#] (36)   [*,#,#,*,+] → [B,#,#,#,A] (24)   [-,B,#,#,#] → [B,+,-,#,#] (11)
[A,A,#,B,#] → [A,+,*,*,*] (35)   [C,#,#,#,B] → [B,#,#,*,+] (16)   [+,B,#,#,#] → [B,+,-,-,#] (33)
[+,A,#,#,#] → [A,+,-,-,-] (32)   [B,#,#,B,#] → [B,#,#,+,*] (15)   [A,B,+,#,#] → [B,+,A,#,#] ( 3)
[A,A,+,#,#] → [A,+,A,#,#] ( 3)   [-,#,#,B,#] → [B,#,#,+,-] (11)   [A,B,+,#,+] → [B,+,A,#,A] ( 7)
[A,A,+,#,+] → [A,+,A,#,A] ( 7)   [A,#,#,B,+] → [B,#,#,+,A] ( 3)   [A,B,+,+,#] → [B,+,A,A,#] ( 5)
[#,A,+,-,#] → [A,+,A,-,#] (83)   [B,#,#,B,+] → [B,#,#,+,B] ( 3)   [A,B,+,+,+] → [B,+,A,A,A] ( 8)
[A,A,+,+,#] → [A,+,A,A,#] ( 5)   [C,#,#,B,+] → [B,#,#,+,C] ( 3)   [B,B,+,#,#] → [B,+,B,#,#] ( 3)
[A,A,+,+,+] → [A,+,A,A,A] ( 8)   [#,#,D,B,+] → [B,#,#,+,D] (91)   [B,B,+,#,+] → [B,+,B,#,B] ( 7)
[B,A,+,#,#] → [A,+,B,#,#] ( 3)   [*,#,*,+,#] → [B,#,#,A,#] (24)   [B,B,+,+,#] → [B,+,B,B,#] ( 5)
[B,A,+,#,+] → [A,+,B,#,B] ( 7)   [A,#,#,+,B] → [B,#,#,A,+] ( 4)   [B,B,+,+,+] → [B,+,B,B,B] ( 8)
[B,A,+,+,#] → [A,+,B,B,#] ( 5)   [*,#,*,+,+] → [B,#,#,A,B] (42)   [C,B,+,#,#] → [B,+,C,#,#] ( 3)
[B,A,+,+,+] → [A,+,B,B,B] ( 8)   [B,#,#,+,B] → [B,#,#,B,+] ( 4)   [C,B,+,#,+] → [B,+,C,#,C] ( 7)
[C,A,+,#,#] → [A,+,C,#,#] ( 3)   [C,#,#,+,B] → [B,#,#,C,+] ( 4)   [C,B,+,+,#] → [B,+,C,C,#] ( 5)
[C,A,+,#,+] → [A,+,C,#,C] ( 7)   [#,D,#,+,C] → [B,#,#,D,+] (92)   [C,B,+,+,+] → [B,+,C,C,C] ( 8)
[C,A,+,+,#] → [A,+,C,C,#] ( 5)   [A,#,#,#,B] → [B,#,*,#,+] (14)   [#,B,+,#,D] → [B,+,D,#,#] (91)
[C,A,+,+,+] → [A,+,C,C,C] ( 8)   [C,#,#,B,#] → [B,#,*,+,#] (16)   [-,#,#,#,B] → [B,-,#,#,+] (11)
[-,#,#,A,#] → [A,-,#,+,#] (10)   [A,#,B,#,#] → [B,#,+,#,*] (14)   [+,#,#,B,#] → [B,-,#,+,-] (33)
[#,-,#,A,+] → [A,-,#,+,A] (83)   [A,#,B,#,+] → [B,#,+,#,A] ( 2)   [+,#,#,#,B] → [B,-,-,#,+] (33)
[+,#,A,#,#] → [A,-,+,-,-] (32)   [B,#,B,#,+] → [B,#,+,#,B] ( 2)   [*,+,#,#,*] → [B,A,#,#,#] (24)
[+,#,#,A,#] → [A,-,-,+,-] (32)   [C,#,B,#,+] → [B,#,+,#,C] ( 2)   [A,+,#,#,B] → [B,A,#,#,+] ( 3)
[+,#,#,#,A] → [A,-,-,-,+] (32)   [D,#,B,#,+] → [B,#,+,#,D] (56)   [A,+,#,B,#] → [B,A,#,+,#] ( 2)
[#,-,+,A,#] → [A,-,A,+,#] (84)   [B,#,B,#,#] → [B,#,+,*,#] (15)   [A,+,#,B,+] → [B,A,#,+,A] ( 5)
[*,+,#,*,#] → [A,A,#,#,#] (23)   [-,#,B,#,#] → [B,#,+,-,#] (11)   [A,+,#,+,B] → [B,A,#,A,+] ( 7)
[A,+,#,#,A] → [A,A,#,#,+] ( 3)   [+,#,B,#,#] → [B,#,+,-,-] (33)   [A,+,B,#,#] → [B,A,+,#,#] ( 4)
[A,+,#,A,#] → [A,A,#,+,#] ( 2)   [A,#,B,+,#] → [B,#,+,A,#] ( 3)   [A,+,B,#,+] → [B,A,+,#,A] ( 6)
[A,+,#,A,+] → [A,A,#,+,A] ( 5)   [A,#,B,+,+] → [B,#,+,A,A] ( 5)   [A,+,B,+,#] → [B,A,+,A,#] ( 7)
[A,+,#,+,A] → [A,A,#,A,+] ( 7)   [B,#,B,+,#] → [B,#,+,B,#] ( 3)   [A,+,B,+,+] → [B,A,+,A,A] ( 8)
[A,+,A,#,#] → [A,A,+,#,#] ( 4)   [B,#,B,+,+] → [B,#,+,B,B] ( 5)   [A,+,+,#,B] → [B,A,A,#,+] ( 5)
[#,+,A,#,-] → [A,A,+,#,-] (84)   [C,#,B,+,#] → [B,#,+,C,#] ( 3)   [A,+,+,B,#] → [B,A,A,+,#] ( 6)
[A,+,A,#,+] → [A,A,+,#,A] ( 6)   [C,#,B,+,+] → [B,#,+,C,C] ( 5)   [A,+,+,B,+] → [B,A,A,+,A] ( 8)
[A,+,A,+,#] → [A,A,+,A,#] ( 7)   [#,D,B,+,#] → [B,#,+,D,#] (91)   [A,+,+,+,B] → [B,A,A,A,+] ( 8)
[A,+,A,+,+] → [A,A,+,A,A] ( 8)   [*,*,+,#,#] → [B,#,A,#,#] (24)   [*,+,+,#,*] → [B,A,B,#,#] (42)
[#,+,-,#,A] → [A,A,-,#,+] (83)   [A,#,+,#,B] → [B,#,A,#,+] ( 2)   [B,+,#,#,B] → [B,B,#,#,+] ( 3)
[A,+,+,#,A] → [A,A,A,#,+] ( 5)   [A,#,+,B,#] → [B,#,A,+,#] ( 4)   [*,+,#,*,+] → [B,B,#,#,A] (42)
[A,+,+,A,#] → [A,A,A,+,#] ( 6)   [A,#,+,B,+] → [B,#,A,+,A] ( 7)   [B,+,#,B,#] → [B,B,#,+,#] ( 2)
[A,+,+,A,+] → [A,A,A,+,A] ( 8)   [A,#,+,+,B] → [B,#,A,A,+] ( 6)   [B,+,#,B,+] → [B,B,#,+,B] ( 5)
[A,+,+,+,A] → [A,A,A,A,+] ( 8)   [*,*,+,+,#] → [B,#,A,B,#] (42)   [B,+,#,+,B] → [B,B,#,B,+] ( 7)
[B,+,#,#,A] → [A,B,#,#,+] ( 3)   [B,#,+,#,B] → [B,#,B,#,+] ( 2)   [B,+,B,#,#] → [B,B,+,#,#] ( 4)
[B,+,#,A,#] → [A,B,#,+,#] ( 2)   [B,#,+,B,#] → [B,#,B,+,#] ( 4)   [B,+,B,#,+] → [B,B,+,#,B] ( 6)
[B,+,#,A,+] → [A,B,#,+,B] ( 5)   [B,#,+,B,+] → [B,#,B,+,B] ( 7)   [B,+,B,+,#] → [B,B,+,B,#] ( 7)
[B,+,#,+,A] → [A,B,#,B,+] ( 7)   [B,#,+,+,B] → [B,#,B,B,+] ( 6)   [B,+,B,+,+] → [B,B,+,B,B] ( 8)
[*,+,*,+,+] → [A,B,#,B,B] (41)   [C,#,+,#,B] → [B,#,C,#,+] ( 2)   [B,+,+,#,B] → [B,B,B,#,+] ( 5)
[B,+,A,#,#] → [A,B,+,#,#] ( 4)   [C,#,+,B,#] → [B,#,C,+,#] ( 4)   [B,+,+,B,#] → [B,B,B,+,#] ( 6)
[B,+,A,#,+] → [A,B,+,#,B] ( 6)   [C,#,+,B,+] → [B,#,C,+,C] ( 7)   [B,+,+,B,+] → [B,B,B,+,B] ( 8)
[B,+,A,+,#] → [A,B,+,B,#] ( 7)   [C,#,+,+,B] → [B,#,C,C,+] ( 6)   [B,+,+,+,B] → [B,B,B,B,+] ( 8)
[B,+,A,+,+] → [A,B,+,B,B] ( 8)   [D,#,+,#,B] → [B,#,D,#,+] (56)   [C,+,#,#,B] → [B,C,#,#,+] ( 3)
[B,+,+,#,A] → [A,B,B,#,+] ( 5)   [#,#,+,C,D] → [B,#,D,+,#] (92)   [C,+,#,B,#] → [B,C,#,+,#] ( 2)
[*,+,+,*,+] → [A,B,B,#,B] (41)   [B,#,#,#,B] → [B,*,#,#,+] (15)   [C,+,#,B,+] → [B,C,#,+,C] ( 5)
[B,+,+,A,#] → [A,B,B,+,#] ( 6)   [A,#,#,B,#] → [B,*,#,+,#] (14)   [C,+,#,+,B] → [B,C,#,C,+] ( 7)
[B,+,+,A,+] → [A,B,B,+,B] ( 8)   [C,#,B,#,#] → [B,*,+,#,#] (16)   [C,+,B,#,#] → [B,C,+,#,#] ( 4)
[*,+,+,+,*] → [A,B,B,B,#] (41)   [C,B,#,#,#] → [B,+,#,#,*] (16)   [C,+,B,#,+] → [B,C,+,#,C] ( 6)
[B,+,+,+,A] → [A,B,B,B,+] ( 8)   [A,B,#,#,+] → [B,+,#,#,A] ( 4)   [C,+,B,+,#] → [B,C,+,C,#] ( 7)
[C,+,#,#,A] → [A,C,#,#,+] ( 3)   [B,B,#,#,+] → [B,+,#,#,B] ( 4)   [C,+,B,+,+] → [B,C,+,C,C] ( 8)
[C,+,#,A,#] → [A,C,#,+,#] ( 2)   [C,B,#,#,+] → [B,+,#,#,C] ( 4)   [C,+,+,#,B] → [B,C,C,#,+] ( 5)
[C,+,#,A,+] → [A,C,#,+,C] ( 5)   [#,C,D,#,+] → [B,+,#,#,D] (92)   [C,+,+,B,#] → [B,C,C,+,#] ( 6)
[C,+,#,+,A] → [A,C,#,C,+] ( 7)   [A,B,#,#,#] → [B,+,#,*,#] (14)   [C,+,+,B,+] → [B,C,C,+,C] ( 8)
[C,+,A,#,#] → [A,C,+,#,#] ( 4)   [A,B,#,+,#] → [B,+,#,A,#] ( 2)   [C,+,+,+,B] → [B,C,C,C,+] ( 8)
[C,+,A,#,+] → [A,C,+,#,C] ( 6)   [A,B,#,+,+] → [B,+,#,A,A] ( 6)   [#,+,#,D,B] → [B,D,#,#,+] (91)
[C,+,A,+,#] → [A,C,+,C,#] ( 7)   [B,B,#,+,#] → [B,+,#,B,#] ( 2)   [D,+,#,B,#] → [B,D,#,+,#] (56)
[C,+,A,+,+] → [A,C,+,C,C] ( 8)   [B,B,#,+,+] → [B,+,#,B,B] ( 6)   [#,+,C,D,#] → [B,D,+,#,#] (92)
[C,+,+,#,A] → [A,C,C,#,+] ( 5)   [C,B,#,+,#] → [B,+,#,C,#] ( 2)   [*,*,#,#,+] → [C,#,#,#,A] (25)
[C,+,+,A,#] → [A,C,C,+,#] ( 6)   [C,B,#,+,+] → [B,+,#,C,C] ( 6)   [C,#,#,#,C] → [C,#,#,*,+] (16)
[C,+,+,A,+] → [A,C,C,+,C] ( 8)   [D,B,#,+,#] → [B,+,#,D,#] (56)   [B,#,#,C,#] → [C,#,#,+,*] (15)
```

Fig. 35. (Continued)

```
[A,#,#,C,+] → [C,#,#,+,A] ( 3)
[B,#,#,C,+] → [C,#,#,+,B] ( 3)
[C,#,#,C,+] → [C,#,#,+,C] ( 3)
[-,#,#,A,+] → [C,#,#,+,D] (75)
[-,#,#,#,C] → [C,#,#,-,+] (12)
[*,#,#,+,*] → [C,#,#,A,#] (25)
[A,#,#,+,C] → [C,#,#,A,+] ( 4)
[B,#,#,+,C] → [C,#,#,B,+] ( 4)
[*,*,#,+,+] → [C,#,#,B,A] (43)
[C,#,#,+,C] → [C,#,#,C,+] ( 4)
[-,#,#,+,A] → [C,#,#,D,+] (76)
[A,#,#,#,C] → [C,#,*,#,+] (14)
[C,#,#,C,#] → [C,#,*,+,#] (16)
[A,#,C,#,#] → [C,#,+,#,*] (14)
[A,#,C,#,+] → [C,#,+,#,A] ( 2)
[B,#,C,#,+] → [C,#,+,#,B] ( 2)
[C,#,C,#,+] → [C,#,+,#,C] ( 2)
[-,#,A,#,+] → [C,#,+,#,D] (74)
[B,#,C,#,#] → [C,#,+,*,#] (15)
[A,#,C,+,#] → [C,#,+,A,#] ( 3)
[A,#,C,+,+] → [C,#,+,A,A] ( 5)
[B,#,C,+,#] → [C,#,+,B,#] ( 3)
[B,#,C,+,+] → [C,#,+,B,B] ( 5)
[C,#,C,+,#] → [C,#,+,C,#] ( 3)
[C,#,C,+,+] → [C,#,+,C,C] ( 5)
[-,#,A,+,#] → [C,#,+,D,#] (75)
[-,#,#,C,#] → [C,#,-,+,#] (12)
[+,#,#,#,C] → [C,#,-,-,+] (34)
[*,#,+,*,#] → [C,#,A,#,#] (25)
[A,#,+,#,C] → [C,#,A,#,+] ( 2)
[A,#,+,C,#] → [C,#,A,+,#] ( 4)
[A,#,+,C,+] → [C,#,A,+,A] ( 7)
[A,#,+,+,C] → [C,#,A,A,+] ( 6)
[B,#,+,#,C] → [C,#,B,#,+] ( 2)
[B,#,+,C,#] → [C,#,B,+,#] ( 4)
[B,#,+,C,+] → [C,#,B,+,B] ( 7)
[*,#,+,+,*] → [C,#,B,A,#] (43)
[B,#,+,+,C] → [C,#,B,B,+] ( 6)
[C,#,+,#,C] → [C,#,C,#,+] ( 2)
[C,#,+,C,#] → [C,#,C,+,#] ( 4)
[C,#,+,C,+] → [C,#,C,+,C] ( 7)
[C,#,+,+,C] → [C,#,C,C,+] ( 6)
[-,#,+,#,A] → [C,#,D,#,+] (74)
[-,#,+,A,#] → [C,#,D,+,#] (76)
[B,#,#,#,C] → [C,*,#,#,+] (15)
[A,#,#,C,#] → [C,*,#,+,#] (14)
[C,#,C,#,#] → [C,*,+,#,#] (16)
[C,C,#,#,#] → [C,+,#,#,*] (16)
[-,C,#,#,#] → [C,+,#,#,-] (12)
[A,C,#,#,+] → [C,+,#,#,A] ( 4)
[B,C,#,#,+] → [C,+,#,#,B] ( 4)
[C,C,#,#,+] → [C,+,#,#,C] ( 4)
[-,A,#,#,+] → [C,+,#,#,D] (76)
[A,C,#,#,#] → [C,+,#,*,#] (14)
[+,C,#,#,#] → [C,+,#,-,-] (34)
[A,C,#,+,#] → [C,+,#,A,#] ( 2)
[A,C,#,+,+] → [C,+,#,A,A] ( 6)
[B,C,#,+,#] → [C,+,#,B,#] ( 2)
[B,C,#,+,+] → [C,+,#,B,B] ( 6)
[C,C,#,+,#] → [C,+,#,C,#] ( 2)
[C,C,#,+,+] → [C,+,#,C,C] ( 6)
[-,A,#,+,#] → [C,+,#,D,#] (74)
[B,C,#,#,#] → [C,+,*,#,#] (15)
[A,C,+,#,#] → [C,+,A,#,#] ( 3)

[A,C,+,#,+] → [C,+,A,#,A] ( 7)
[A,C,+,+,#] → [C,+,A,A,#] ( 5)
[A,C,+,+,+] → [C,+,A,A,A] ( 8)
[B,C,+,#,#] → [C,+,B,#,#] ( 3)
[B,C,+,#,+] → [C,+,B,#,B] ( 7)
[B,C,+,+,#] → [C,+,B,B,#] ( 5)
[B,C,+,+,+] → [C,+,B,B,B] ( 8)
[C,C,+,#,#] → [C,+,C,#,#] ( 3)
[C,C,+,#,+] → [C,+,C,#,C] ( 7)
[C,C,+,+,#] → [C,+,C,C,#] ( 5)
[C,C,+,+,+] → [C,+,C,C,C] ( 8)
[-,A,+,#,#] → [C,+,D,#,#] (75)
[-,#,C,#,#] → [C,-,+,#,#] (12)
[+,#,C,#,#] → [C,-,+,#,-] (34)
[+,#,#,C,#] → [C,-,-,+,#] (34)
[*,+,*,#,#] → [C,A,#,#,#] (25)
[A,+,#,#,C] → [C,A,#,#,+] ( 3)
[*,+,*,#,+] → [C,A,#,#,B] (43)
[A,+,#,C,#] → [C,A,#,+,#] ( 2)
[A,+,#,C,+] → [C,A,#,+,A] ( 5)
[A,+,#,+,C] → [C,A,#,A,+] ( 7)
[A,+,C,#,#] → [C,A,+,#,#] ( 4)
[A,+,C,#,+] → [C,A,+,#,A] ( 6)
[A,+,C,+,#] → [C,A,+,A,#] ( 7)
[A,+,C,+,+] → [C,A,+,A,A] ( 8)
[A,+,+,#,C] → [C,A,A,#,+] ( 5)
[A,+,+,C,#] → [C,A,A,+,#] ( 6)
[A,+,+,C,+] → [C,A,A,+,A] ( 8)
[A,+,+,+,C] → [C,A,A,A,+] ( 8)
[B,+,#,#,C] → [C,B,#,#,+] ( 3)
[B,+,#,C,#] → [C,B,#,+,#] ( 2)
[B,+,#,C,+] → [C,B,#,+,B] ( 5)
[B,+,#,+,C] → [C,B,#,B,+] ( 7)
[B,+,C,#,#] → [C,B,+,#,#] ( 4)
[B,+,C,#,+] → [C,B,+,#,B] ( 6)
[B,+,C,+,#] → [C,B,+,B,#] ( 7)
[B,+,C,+,+] → [C,B,+,B,B] ( 8)
[*,+,+,*,#] → [C,B,A,#,#] (43)
[B,+,+,#,C] → [C,B,B,#,+] ( 5)
[B,+,+,C,#] → [C,B,B,+,#] ( 6)
[B,+,+,C,+] → [C,B,B,+,B] ( 8)
[B,+,+,+,C] → [C,B,B,B,+] ( 8)
[C,+,#,#,C] → [C,C,#,#,+] ( 3)
[C,+,#,C,#] → [C,C,#,+,#] ( 2)
[C,+,#,C,+] → [C,C,#,+,C] ( 5)
[C,+,#,+,C] → [C,C,#,C,+] ( 7)
[C,+,C,#,#] → [C,C,+,#,#] ( 4)
[C,+,C,#,+] → [C,C,+,#,C] ( 6)
[C,+,C,+,#] → [C,C,+,C,#] ( 7)
[C,+,C,+,+] → [C,C,+,C,C] ( 8)
[C,+,+,#,C] → [C,C,C,#,+] ( 5)
[C,+,+,C,#] → [C,C,C,+,#] ( 6)
[C,+,+,C,+] → [C,C,C,+,C] ( 8)
[C,+,+,+,C] → [C,C,C,C,+] ( 8)
[-,+,#,#,A] → [C,D,#,#,+] (75)
[-,+,#,A,#] → [C,D,#,+,#] (74)
[-,+,A,#,#] → [C,D,+,#,#] (76)
[A,#,#,D,+] → [D,#,#,+,A] (54)
[B,#,#,D,+] → [D,#,#,+,B] (54)
[C,#,#,D,+] → [D,#,#,+,C] (54)
[A,#,#,+,D] → [D,#,#,A,+] (55)
[B,#,#,+,D] → [D,#,#,B,+] (55)
[C,#,#,+,D] → [D,#,#,C,+] (55)
[D,#,+,+,C] → [D,#,*,A,+] (80)

[A,#,D,#,+] → [D,#,+,#,A] (53)
[B,#,D,#,+] → [D,#,+,#,B] (53)
[C,#,D,#,+] → [D,#,+,#,C] (53)
[A,#,D,+,#] → [D,#,+,A,#] (54)
[D,#,C,+,+] → [D,#,+,A,*] (79)
[*,#,D,+,+] → [D,#,+,A,A] (77)
[-,#,D,+,+] → [D,#,+,A,B] (77)
[+,#,D,+,+] → [D,#,+,A,C] (77)
[B,#,D,+,#] → [D,#,+,B,#] (54)
[C,#,D,+,#] → [D,#,+,C,#] (54)
[A,#,+,#,D] → [D,#,A,#,+] (53)
[A,#,+,D,#] → [D,#,A,+,#] (55)
[*,#,+,+,D] → [D,#,A,A,+] (78)
[B,#,+,#,D] → [D,#,B,#,+] (53)
[B,#,+,D,#] → [D,#,B,+,#] (55)
[-,#,+,+,D] → [D,#,B,A,+] (78)
[C,#,+,#,D] → [D,#,C,#,+] (53)
[C,#,+,D,#] → [D,#,C,+,#] (55)
[+,#,+,+,D] → [D,#,C,A,+] (78)
[D,+,#,C,+] → [D,*,#,+,A] (79)
[D,+,+,C,#] → [D,*,A,+,#] (80)
[A,D,#,#,+] → [D,+,#,#,A] (55)
[B,D,#,#,+] → [D,+,#,#,B] (55)
[C,D,#,#,+] → [D,+,#,#,C] (55)
[D,C,#,+,+] → [D,+,#,*,A] (80)
[A,D,#,+,#] → [D,+,#,A,#] (53)
[*,D,#,+,+] → [D,+,#,A,A] (78)
[B,D,#,+,#] → [D,+,#,B,#] (53)
[-,D,#,+,+] → [D,+,#,B,A] (78)
[C,D,#,+,#] → [D,+,#,C,#] (53)
[+,D,#,+,+] → [D,+,#,C,A] (78)
[A,D,+,#,#] → [D,+,A,#,#] (54)
[D,C,+,+,#] → [D,+,A,*,#] (79)
[*,D,+,+,#] → [D,+,A,A,#] (77)
[-,D,+,+,#] → [D,+,A,B,#] (77)
[+,D,+,+,#] → [D,+,A,C,#] (77)
[B,D,+,#,#] → [D,+,B,#,#] (54)
[C,D,+,#,#] → [D,+,C,#,#] (54)
[A,+,#,#,D] → [D,A,#,#,+] (54)
[A,+,#,D,#] → [D,A,#,+,#] (53)
[*,+,#,D,+] → [D,A,#,+,A] (77)
[D,+,+,#,C] → [D,A,*,#,+] (79)
[A,+,D,#,#] → [D,A,+,#,#] (55)
[D,+,C,#,+] → [D,A,+,#,*] (80)
[*,+,D,#,+] → [D,A,+,#,A] (78)
[-,+,D,#,+] → [D,A,+,#,B] (78)
[+,+,D,#,+] → [D,A,+,#,C] (78)
[*,+,+,#,D] → [D,A,A,#,+] (77)
[*,+,+,D,#] → [D,A,A,+,#] (78)
[-,+,+,#,D] → [D,A,B,#,+] (77)
[+,+,+,#,D] → [D,A,C,#,+] (77)
[B,+,#,#,D] → [D,B,#,#,+] (54)
[B,+,#,D,#] → [D,B,#,+,#] (53)
[-,+,#,D,+] → [D,B,#,+,A] (77)
[B,+,D,#,#] → [D,B,+,#,#] (55)
[-,+,+,D,#] → [D,B,A,+,#] (78)
[C,+,#,#,D] → [D,C,#,#,+] (54)
[C,+,#,D,#] → [D,C,#,+,#] (53)
[+,+,#,D,+] → [D,C,#,+,A] (77)
[C,+,D,#,#] → [D,C,+,#,#] (55)
[+,+,+,D,#] → [D,C,A,+,#] (78)
```

Fig. 35. (Continued)

Acknowledgements

The authors express their thanks to the referee for his valuable comments.

References

[1] C.H. Bennett, Logical reversibility of computation, *IBM J. Res. Dev.* **17** (1973) 525–532.
[2] E.F. Codd, *Cellular Automata* (Academic Press, New York, 1968).
[3] C.G. Langton, Self-reproduction in cellular automata, *Physica* **D10** (1984) 135–144.
[4] N. Margolus, Physics-like model of computation, *Physica* **D10** (1984) 81–95.
[5] K. Morita, Computation universality of one-dimensional one-way reversible cellular automata, *Inform. Process. Lett.* **42** (1992) 325–329.
[6] K. Morita, Reversible simulation of one-dimensional irreversible cellular automata, *Theoret. Comput. Sci.* **148** (1995) 157–163.
[7] K. Morita and M. Harao, Computation universality of one-dimensional reversible (injective) cellular automata, *Trans. IEICE Japan* **E72** (1989) 758–762.
[8] K. Morita and S. Ueno, Computation-universal models of two-dimensional 16-state reversible cellular automata, *IEICE Trans. Inform. Systems* **E75-D** (1992) 141–147.
[9] J. von Neumann, *Theory of Self-reproducing Automata* A.W. Burks, ed. (The University of Illinois Press, Urbana, 1966).
[10] T. Toffoli, Computation and construction universality of reversible cellular automata, *J. Comput. Systems Sci.* **15** (1977) 213–231.
[11] T. Toffoli and N. Margolus, Invertible cellular automata: a review, *Physica D* **45** (1990) 229–253.