27716

Computation-Universality, Synchronization and Self-Reproduction in Reversible and Conservative Cellular Automata

Katsunobu IMAI

December 1998

Computation-Universality, Synchronization and Self-Reproduction in Reversible and Conservative Cellular Automata

Katsunobu IMAI

December 1998

内容梗概

単電子デバイス，量子計算，DNA 計算などの次世代の計算機構と目されているものの多くは，情報がキャリアとなる物質に直接結びついており，従来の電子デバイスとは異なる演算の枠組みが求められている．また，従来の MOS デバイスにおいても，集積化にともない 1 ビットのキャリアとしての電子数が減少し，従来用いられているような論理回路の構成は困難になると言われている．そのためキャリアとしての物質の物理的性質を反映した計算モデルの研究が注目されるようになってきている．その中でも可逆論理に基づく可逆計算モデルが近年注目を集めるようになってきた．たとえば量子計算においては計算過程が可逆であり，観測時を除いてどの計算のステージも情報の欠損を伴わず，その計算過程は可逆的なものとして構成されなければならない．

そのような可逆な計算システムのモデルのひとつとして，可逆セル・オートマトン (CA) がある．可逆 CA とは，どの時点でも必ず 1 ステップ前の状態に一意に逆戻りできるような CA である．この性質のため，可逆 CA 上での信号やパターンの生成や消滅には制約が伴うことになるが，それにもかかわらず一般の CA 同様，計算万能性を有するものが知られている．しかし，体系的な研究はあまり多くはなされていない．さらに従来の CA では多数の研究が行われている構成万能性や同期問題など，より高次の機能についてはほとんど調べられていなかった．同期動作の実現には相互のセルの通信が不可欠であるが，可逆 CA 上では，自由に信号を消滅させられないという制約から一般的にはごみ情報と呼ばれる不要な信号を散逸させてしまう．そのため複数のセルの同期動作を実現するためには従来の CA 上とは異なるアプローチが必要であると考えられる．このことは量子計算モデルなどの可逆計算過程において計算の初期状態を同時に（同期をとって）配置し，計算を開始することが困難である問題に対応すると考えられている．

そこで本論文では，可逆 CA における計算能力の研究を，計算万能性，同期動作，パターン配置問題としての自己増殖能力の観点から行った．まず計算万能性について考察し，従来から知られていたモデルよりも状態数の少ない 2 次元の計算万能可逆 CA を示した．次に従来の非可逆な CA の同期問題として研究されてきた一斉射撃問題を可逆 CA 上で実現するための条件を示し，実際に可逆解を構成した．さらに質量保存的な性質を反映した CA によるモデルとしてビット保存的な CA を拡張した Number-Conserving CA を提案し，その上での一斉射撃解を構成した．また，可逆システムにおける初期状態配置の問題に対するアプローチとして，2 次元と 3 次元可逆空間における簡単メカニズムに基づく自己増殖セル・オートマトンが構成可能であることを示した．

第1章，第2章において，研究背景ならびに，可逆CAについての諸定義や既知の主要な結果について述べる．

第3章では計算万能性を有する8状態の可逆CAを構築した結果を示す．まず可逆セル・オートマトンの計算万能性について，その定式化に用いられている保存論理とともに概観する．計算万能な可逆セル・オートマトンとしては，Margolusによるモデルや Morita, Uenoによる分割セル・オートマトン (PCA) を用いたモデルが知られている．Margolusのモデルは特殊な近傍を持つモデルであったため，Morita と Uenoは分割したパートを持つものの従来のCAのサブクラスと見なすことができるPCAを用いて16状態のモデルを構成した．しかし，本章では3角形状の近傍を用いることにより8状態の計算万能なモデルを構成し，状態数がさらに削減可能であることを示す．これはPCAの枠組みを用いた等方的な規則を持つモデルの中では最小状態数のものである．まず16状態の三角形状計算万能モデルを構成した結果を示し，それを8状態に縮小する．構成したCAが計算万能であることは，万能な可逆論理素子である Fredkin ゲートの入出力関係を模倣できることによって示される．

第4章では，CA上の同期問題の基本的なモデルとして用いられている一斉射撃問題をとりあげ，可逆CAや Number-Conserving CA における実現可能性を議論する．まず従来の非保存的なセル・オートマトンの上の一斉射撃問題について概説し，一斉射撃問題を保存的なCAで実現するための条件について議論する．ここでは従来の単一の射撃状態からなる可逆解は構成できないことを示し，可逆CAにおける一斉射撃解条件を新たに定義する．この条件の下で，Minsky らの $3n$ 時間解に基づく99状態解を1次元可逆PCAによって構成した結果を示す．さらに2次元の場合の解や，より高速に同期する解などのバリエーションについての結果も示す．次に Number-Conserving CA 上における一斉射撃問題を考察する．まず Number-Conserving CA の基本的な性質について考察し，その上での一斉射撃解の条件について述べる．そして，実際に Number-Conserving CA 上の解が構成可能であることを示す．

第5章では，2次元可逆空間で自己増殖する 8^5 状態のセル・オートマトン SR_8 を構成する．これは Langton の8状態自己増殖CAに基づいているが，自分自身の形状を参照することにより，簡単な機構にもかかわらず自由度の高い増殖メカニズムを可逆空間上に実現している．また，SR_8 の自己参照メカニズムは増殖時に形状を子孫へ伝えるために用いられるが，さらにパターンの配置場所の制御にも応用できることを述べる．さらに，SR_8 を基本的な性質を保ったままで3次元へ拡張できることを示し，可逆空間におけるパターン配置のための枠組みとなりうることを示す．

Acknowledgements

This work was produced with co-workers Kenichi Morita, Masanori Ogino, Taichi Adachi, Katsunori Chikamune, Shinichi Furusaka, Takahiro Hori, Kenji Sako, Yukio Matsuda and Kenji Fujita.

I wish to express my gratitude to Shunsuke Sato, Kaoru Nakano and Kunihiko Fukushima at Osaka University for their useful comments and suggestions as supervisers.

I would thank Jaques Mazoyer at ENS Lyon, Maurice Margenstern at I.U.T. de Metz and Hiroyuki Aga at SONY Co, Ltd. for their useful comments and disscusions.

I would also thank Chuzo Iwamoto and all members of Knowledge Engineering Laboratory for their support.

Almost all simulations are performed by Macintosh Common Lisp and its graphic libraries. These systems are maintained by engineers at Digitool, Inc. and John Wiseman at Neodesic, Inc.

Contents

1 Introduction **1**

2 Cellular Automata **5**

2.1 Definitions 5

2.2 Computation- and Construction-Universality of CA 7

2.3 Firing Squad Synchronization Problem 9

2.3.1 $3n$-time Solutions 10

2.3.2 Minimal Time Solutions 11

2.3.3 Higher-Dimensional Cases 12

2.3.4 Standard Synchronizing Condition (SSC) 12

2.4 Reversible and Conservative Cellular Automata 13

2.4.1 CA and Conservative Constraints 13

2.4.2 Margolus Neighborhood 14

2.4.3 Partitioned CA (PCA) 16

3 Computation-Universal Two-Dimensional Triangular Reversible CA **23**

3.1 Definition of Triangular PCA 23

3.2 16-state Triangular RPCA 25

3.3 8-state Triangular RPCA 27

3.3.1 A Computation-Universal 8-state Model 28

4 Synchronization on Reversible and Conservative Cellular Automata **37**

4.1 Reversible Solutions for FSSP 37

4.1.1 FSSP Conditions for Reversible Solutions 38

4.1.2 $3n$-time Solutions 40

4.1.3 2^k-divided Solutions on Reversible PCA 43

4.1.4 Minimal Time Solutions and Reversibility 50

4.1.5 Two-dimensional Cases 58
4.1.6 Summary 69
4.2 Number Conserving Solutions for FSSP 69
4.2.1 Number-Conserving Partitioned Cellular Automata 69
4.2.2 FSSP Conditions for Number-Conserving Solutions 70
4.2.3 A Solution on Number-Conserving PCA 71

5 Self-Reproduction in Reversible Cellular Automata **75**
5.1 Self-Reproduction in a Two-Dimensional RPCA 75
5.1.1 Definition of SR_8 75
5.1.2 Signal Transmission on a Wire 76
5.1.3 A Worm 76
5.1.4 Self-Reproduction of a Worm 78
5.1.5 Self-Reproduction of a Loop 79
5.1.6 Controlling the Position of Daughter Loops in SR_8 79
5.2 Self-Reproduction in a Three-Dimensional RPCA 85
5.2.1 An Extension of SR_8 into a Rotation-Symmeric Three-Dimensional RPCA 85
5.2.2 Three-Dimensional Self-Reproducing RPCA (SR_9) 86
5.2.3 Controlling the Position of Daughter Loops in SR_9 87

6 Conclusion **89**

A Transition rules for $2n+2$-time solution **99**

B Transition rules for SR_8 **103**

C Transition rules for SR_9 **107**

Chapter 1

Introduction

Year by year, the number of transistors contained in a MOS LSI becomes larger and larger, chip size and power dissipation smaller and smaller, and the number of electrons as a carrier of information goes to less and less. And it is said that one-bit must be represented with only dozens of electrons in near future. So many researchers try to find a break-through for this problem and several approaches such as single-electron devices and the adiabatic CMOS [5] are proposed.

Recently several non-electron based computing models, such as quantum computing [17] and DNA computing [2], were proposed and these models turned out to have possibility to achive higher computing efficiency. In such computing models which are considered as computing mechanism of the next generation, information is closely related with substances as carriers. And it is considered that different approaches rather than that of traditional logical devices are needed. For example, in traditional electrical digital circuits, when a bit changes its value, electrons return back to power lines and they are consumed as heat. But this inevitably causes energy dissipations. Then the idea of CMOS are used, and moreover in the adiabatic CMOS model, conservation of information carriers have an important roll [5]. In quantum computing, every computing stages are represented by unitary transformations and it is very difficult to place preferred initial patterns and advance computing using only with unitary transformations [17, 76]. So importance of reversible and conservative computing models come to be widely noticed.

A reversible cellular automaton (RCA) is one of the reversible computing models. Its global function is injective and every configuration has at most one predecessor. Intuitively, it "remembers" the initial configuration and one can reconstruct its initial configuration from a configuration of any time. So reversible property is a strong constraint and one cannot generate nor extinct signals freely. Hence RCAs had been considered not to be computation-universal, i.e., they cannot simulate deterministic universal Turing Machine.

But in late 70's to early 80's, cellular automata are intensively studied again. Toffoli showed that there exist computation-universal RCAs [79] and an important reversible model, the BBM cellular automaton (BBMCA), was introduced by Margolus [47]. It is computation-universal and it has a direct relation with a physical reversible and conservative computing model (the Billiard Ball Model) [21, 19]. The Billiard Ball Model (BBM) has an important aspect that it is possible to compute any function without dissipation of balls as garbages. Once von Neumann conjectured that computing without erasing information is impossible and erasing one-bit infomation must dissipate at least $ln2\ kT$ joules of energy. But the BBM is a conservative model and it is possible to construct a computer that can computes with no energy dissipation in principle. It is also used for hydrodynamic modeling by physicists [18], and the BBMCA is regarded as one of the most important contact point between physical phenomena and analysis of reversible computing.

So in this paper, we have studied reversible and conservative computing processes by RCAs. We show simple RCAs which has computation-universality and we also construct RCAs which have synchronizing ability or self-reproducing ability.

In chapter 3, we discuss how simple RCAs have abilities of universal-computation. We show an 8-state computation-universal RCA model. This model is an improvement of Morita and Ueno's 16-state RCA models [56].

As mentioned above, it is very difficult to place a preferred initial configuration and start computing on reversible and conservative computing models. This problem is described as a restriction of generation and extinction of signals in RCAs, and synchronizing signals and distributing specific patterns on RCAs are also quite difficult. So in chapter 4.1, we discuss synchronization problem of signals in RCAs. We have studied the "Firing squad synchronization problem (FSSP)" in RCAs. This is a traditional synchronization problem for usual CAs and widely studied [84, 9, 42, 48]. We show several solutions to the FSSP on RCAs. Due to the reversibility constraint, some FSSP algorithms can't be used. But $3n$ and $2n+2$-time solution can be constructible. In this chapter, we also introduce the number-conserving property as a conservation constraint and we also construct number-conserving solutions to the FSSP.

In chapter 5, we consider distribution problems of preferred initial patterns on RCAs. We construct a two-dimensional self-reproducing RCA (SR_8) based on a shape-encoding mechanism and this model is an approach to the problem. We also extend it into a three-dimentional RCA.

Chapter 2

Cellular Automata

2.1 Definitions

Definition 2.1.1 A *deterministic one-dimensional cellular automaton* (CA) is a system defined by

$$A = (\mathbf{Z}, Q, N, \varphi_A, \#),$$

where $\mathbf{Z}$ is the set of all integers, Q is a non-empty finite set of internal states of each cell, $N = (m_1, m_2, \cdots, m_k)$ $(m_i \in \mathbf{Z})$ is a neighborhood index, $\varphi_A : Q^k \to Q$ is a mapping called a *local function*, and $\# \in Q$ is a *quiescent state* which satisfies $\varphi_A(\#, \cdots, \#) = \#$. A CA is called a *three-neighbor* CA if $N = (-1, 0, 1)$, and in this case, it can be denoted by $(\mathbf{Z}, Q, \varphi_A, \#)$. A three-neighbor CA is also said to be a CA with *radius* 1, i.e., neighborhood index of a CA with radius r is $(-r, \cdots, 0, \cdots, r)$.

A *configuration* over Q is a mapping $c : \mathbf{Z} \to Q$. Let $\mathrm{Conf}(Q)$ denote the set of all configurations over Q, i.e., $\mathrm{Conf}(Q) = \{c \mid c : \mathbf{Z} \to Q\}$. The function $\Phi_A : \mathrm{Conf}(Q) \to \mathrm{Conf}(Q)$ defined as follows is called the *global function* of A.

$$\Phi_A(c)(i) = \varphi_A(c(i + m_1), c(i + m_2), \cdots, c(i + m_k))$$

We also define a two-dimensional CA. Neighborhood indices of two-dimensional CAs are more complex than one-dimensional cases. So only two neighborhood indices, *von Neumann neighborhood* and *Moore neighborhood* are often used (Fig. 2.1).

Definition 2.1.2 A *deterministic two-dimensional cellular automaton*(CA) with von Neumann neighborhood is defined by

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

where $\mathbf{Z}$ is a set of all integers, Q is a finite set of states of each cells $\varphi_A : Q^5 \to Q$ is a mapping called a local function, and $\varphi_A \in Q$ is a *quiescent state* which satisfies $\varphi_A(\#, \#, \#, \#, \#) = \#$.

A *configuration* over Q is a mapping $c : \mathbf{Z}^2 \to Q$. Let $\mathrm{Conf}(Q)$ denote the set of all configurations over Q, i.e., $\mathrm{Conf}(Q) = \{c | c : \mathbf{Z}^2 \to Q\}$ The function $\Phi_A : \mathrm{Conf}(Q) \to$ Conf(Q) defined as follows is called the *global function* of A.

$$\forall (x, y) \in \mathbf{Z}^2, \Phi_A(c)(x, y) = \varphi_A(c(x, y), c(x, y + 1), c(x + 1, y), c(x, y - 1), c(x - 1, y))$$

A CA with Moore neighborhood is defined in the same way. Its local function is defined as $\varphi_A : Q^9 \to Q$ and its neighborhood index is depicted in Fig. 2.1.

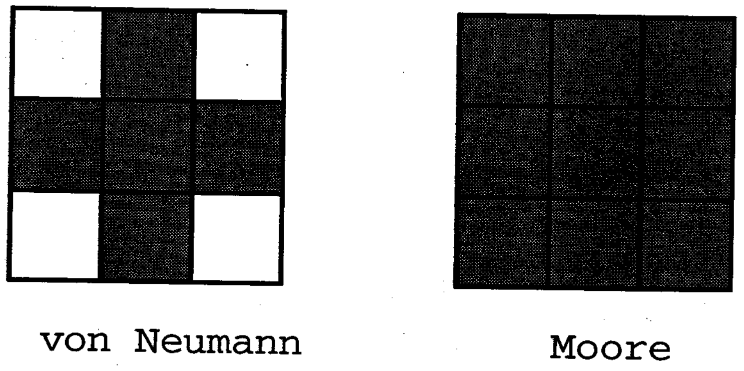


Figure 2.1: Typical neighborhood indices of two-dimensional CA

Throughout this paper, we assume two-dimensional CAs have von Neumann neighborhood unless specified otherwise.

In two-dimensional CAs, there are directions in its cellular spaces. So we can consider following *isotropic* properties.

A is called a *rotation-symmetric* CA iff (i) hold.

(i) $\forall m, a, b, c, d, m' \in Q$

if $\varphi(m, a, b, c, d) = m'$ then $\varphi(m, b, c, d, a) = m'$

A rotation-symmetric CA is also called *reflection-symmetric* iff (ii) holds.

(ii) $\forall m, a, b, c, d, m' \in Q$

if $\varphi(m, a, b, c, d) = m'$ then $\varphi(m, a, d, c, b) = m'$

Intuitively, on a two-dimensional reflection-symmetric CA, the plane has no distinction of between two surfaces. In one-dimensional case, reflection-symmetry and rotation-symmetry are the same notions. Moreover, higher-dimensional cases such as three-dimensional CAs can be defined in the same way. Of course, their isotropic properties become more complex.

2.2 Computation- and Construction-Universality of CA

In this section, we briefly describe the history of computation- and construction-universality of cellular automata.

A deterministic CA A is said to be computation-universal iff A can simulate any deterministic Turing Machines (TM), and A is said to be intrinsically computation-universal iff it is capable of simulating any deterministic CAs of the same dimension. Although intrinsically-universal is stronger notion, we handle only computation-universal CAs through out this paper.

In 1950s, von Neumann showed that his 29-state cellular automaton has an ability to self-reproduction, like a organism [66]. His CA has an ability to simulate a universal TM and thus it was the *first* computation-universal CA.

He used the following strategy to construct his CA.

(A) logical universality

(B) constructibility

(C) construction-universality

(D) self-reproduction

(E) evolution

He used a tape as "gene" for his logical organism, and a universal constructor interpret its tape and construct its copy. His strategy and model were outstanding because the structure of DNA had not been specified in the era.

Anyway he placed computation-universality as the basic property of his logical organism because he want to exclude trivial self-reproduction such as crystal growth. And he regarded his organism as an integration of construction-universality and computation-universality. But von Neumann's model was too complex, so after it was widely known, many smaller state models were reported. Codd [16] showed an 8-state model and Serizawa [73] reported a 3-state model. Codd also proved that any 2-state von Neumann neighbor CA cannot be computation-universal [16], thus 3-state is the smallest in two-dimensional von Neumann neighbor CAs. But increasing the number of neighbors CAs get more ability. In fact, a 2-state Moore neighbor CA can be computation- and construction-universal; an example is the game of Life [23, 13]. The relation between the number of neighborhood size and universality, i.e., the universality of 2-state 6,7,8-neighbor CAs has not been studied by now.

Codd, and Serizawa's CAs are said to be computation- and construction-universal, but notions of construction-universality and self-reproducuction are rather ambiguous than computation-universality. So several researchers began to propose CA models which has only computation-universal or an ability to self-reproduce. Langton thought that computation-universality is not inevitable for self-reproducing organisms and he showed a simple 8-state self-reproducing CA (Fig.2.2) [44]. He posed a criterion only requiring that the self-reproduction should be activly controlled by a mother structure by processing a genetic code of itself in two manners. It is "interpreted" to make a physical shape of daughter structure, and just transferred into the daughter "uninterpreted". It is possible to avoid trivial self-reproduction by this criterion.

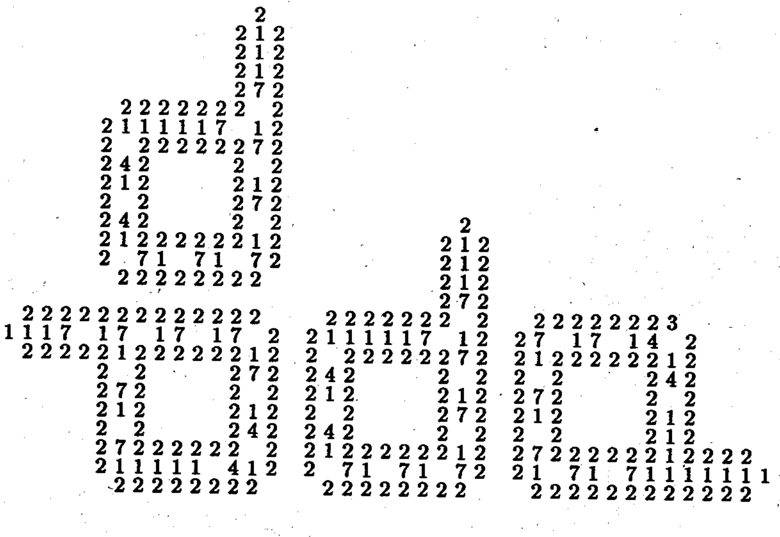


Figure 2.2: Langton's Loop($t = 300$)

After that, several variants and simplified models appeared [14, 70]. For example, Byl [14] showed a 6-state CA model in which a configuration consisting of 10 cells can self-reproduce, reducing both the numbers of states and cells.

Next we consider computation-universality of two-dimensional CAs. Their computation-universality can be proved by embedding universal logic elements. There are several sets of universal logic elements [51]. For example, computation-universality of the game of Life was proved by constructing AND, OR, NOT, and fan-out gates on its cellular space. Sequences of glider patterns were used as signal carriers and glider generators found by Gosper and Speciner et al. [24] act as "batteries" and "clocks". Finally it is proved that the Life can simulate a universal counter machine [13, 68]. Among the 3-state von Neumann neighbor models, Serizawa's model also uses a special type of "spaceship" for signal carriers and one can construct small state computation-universal CAs by using this approach.

In the case of one-dimensional CAs, it is hard to consider to "wire" any logic circuits, and it is impossible to take a constructive approach to prove its computation-universality. Proving universality of a one-dimensional CA, direct simulation of a universal Turing Machine are usually used. As a result, the number of states are apt to enlarge. Currently, 7-state von Neumann neighbor (radius 1) universal CA is known [45]. But even smaller state (such as 3,4,5-state) CAs can generate variety of configurations and Wolfram suggests that the class 4 CAs can be computation-universal. So 4 or 5 state universal CAs might be possible [86]. In one-dimensional CAs, the number of neighbors is also important for its computing ability. 4-state universal CA with radius 2 is known [45].

To the contrary, similar to the situation of proving the determinisity of Turing Machines with small tape-symbols and states, proofs of decidability of small state CAs are very difficult.

2.3 Firing Squad Synchronization Problem

The Firing Squad Synchronization Problem (FSSP) was first devised by Myhill and was introduced by Moore[52]. This is the problem to construct a one-dimensional three-neighbor cellular automata of arbitrary finite length such that one of the end cell (general) makes all the other cells (soldiers) be in a particular state (firing state) at a certain time.

Although Myhill thought that the notion of FSSP is useful for synchronizing all parts of a self-reproducing machine, the FSSP itself has been solely studied by many researchers and many applications are developed by now.

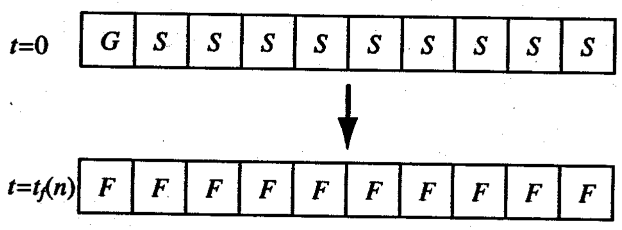


Figure 2.3: Firing Squad Synchronization Problem

2.3.1 $3n$-time Solutions

At first, it was thought that there is no solution to this problem. But it was first solved by Minsky and McCarthy [51]. They constructed a solution synchronizing n cells in $3n$ steps using a divide-and-conquer method. Until now, a 7-state solution based on this algorithm has been known by Yunès [88].

The outline of $3n$-time solution is shown in Fig. 2.4. Reflected signal of velocity 1 meets a signal of velocity $1/3$ at the center cell, and the problem is divided into two sub-problems.

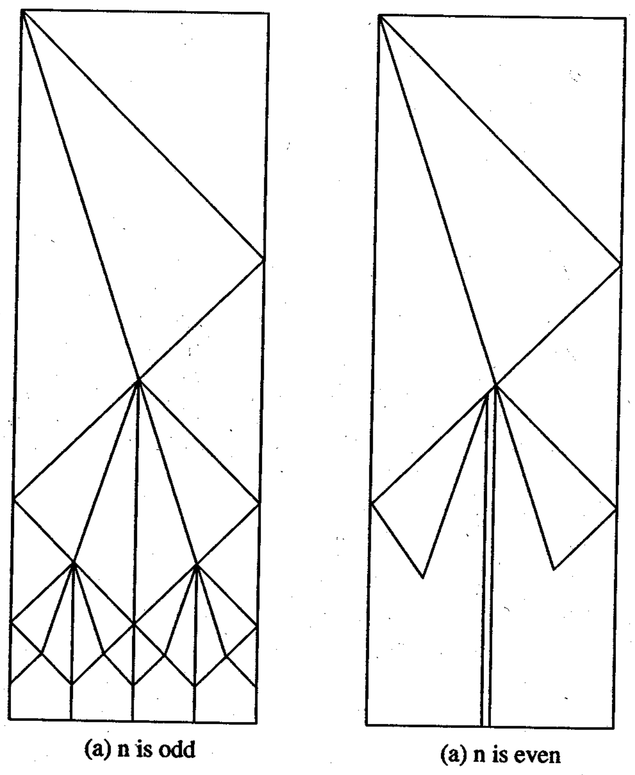


Figure 2.4: Geometries of $3n$-time solution

As depicted in Fig. 2.4, the phase of collisions of velocity -1 and $1/3$ signals varies as the number of cell is even or odd. If the number is odd, two cells are changed into new general cells.

2.3.2 Minimal Time Solutions

Minimal time solutions are completely different from $3n$-time solutions. Goto proved that $2n - 2$ steps is the minimal steps for the FSSP, and he gave a minimal time solution to this problem [29]. Then Waksman [84] and Balzar [9] reduced the number of states. At present 6-state minimal time solution has been given by Mazoyer[48]. The outlines of Waksman-Balzar type and Mazoyer type solutions are depicted in Fig. 2.5.

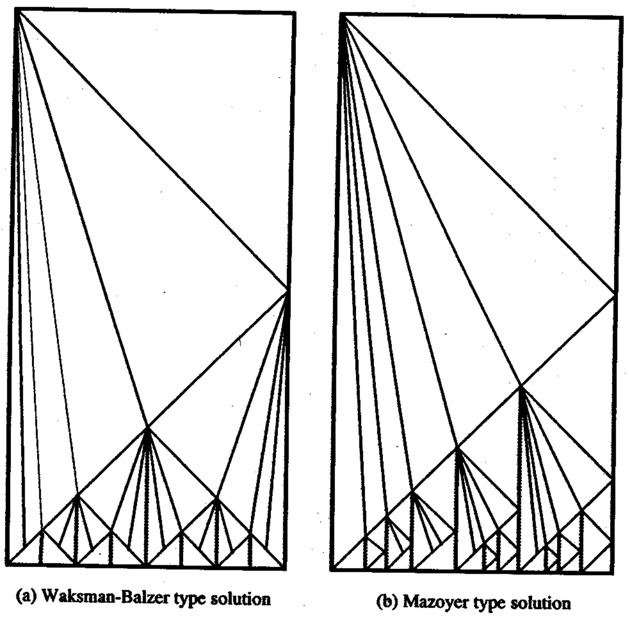


Figure 2.5: Geometries of minimal time solutions

In Waksman-Balzar solution, generals must generate synchronizing signals in both directions. But in Mazoyer's solution, generals generate signals only in one direction, and thus he succeeded to reduce its state. It has been proved that 4-state is insufficient to construct solutions. But existence or inexistence of 5-state solution is still an open problem.

Waksman, Balzar and Mazoyer's solutions use recursive generation of slower signals to synchronize, but Goto's original solution did not uses recursive signal generation. Unfortunately, Goto's solution was not published, and its existence was only known by Moore's comment [52]. Lecture notes [28] are said to exist, but nowadays no researchers have a chance to see it and Goto himself does not have it. There are also a short notes written in Japanese [29] referring to his solution exists. Although it is too short to "decode" his complete algorithm correctly, recently Umeo [83] reconstructed Goto's solution. Further Mazoyer [50] constructed another solution based on the algorithm of Goto's solution. In this paper, Mazoyer's algorithm is used to construct our reversible solution in section 4.1.4.

2.3.3 Higher-Dimensional Cases

In the two-dimensional case, Rosenstiehl [72] showed a solution for any connected figures. Shinkahr [75] also showed that solutions to FSSP on two- and three-dimensional arrays. Some special classes of figures can be synchronized shorter time, and Kobayashi [42] howed some special classes of figures synchronizing in linear time. Nishitani [67] also reported a faster linear time solution. Until now, many variations of FSSPs were studied.

2.3.4 Standard Synchronizing Condition (SSC)

In this section, we define the Firing Squad Synchronization Problem formally, i.e., we define the Synchronizing Condition for one-dimensional three-neighbor CAs. In contrast with traditional model of the FSSP (i.e., n connected automata are considered for firing), we consider an infinite length one-dimensional CA and n automata are embedded into the CA.

Let $A = (\mathbf{Z}, Q, (-1,0,1), \varphi_A, \#)$ be a three-neighbor CA. A standard synchronizing condition SSC for A is stated as follows.

[SSC] There exist three distinct states $g, s, f \in Q - \{\#\}$ that satisfy the following (g, s and f correspond to *general*, *soldier*, and *firing* states, respectively).

1. $\varphi_A(\#, s, s) = s$, $\varphi_A(s, s, s) = s$, and $\varphi_A(s, s, \#) = s$.

2. Let $c_s^{(n)}$ be a configuration defined by

$$c_s^{(n)}(x) = \begin{cases} g & \text{if } x = 1 \\ s & \text{if } x = 2, \cdots, n \\ \# & \text{if } x \le 0 \text{ or } x \ge n+1. \end{cases}$$

Then, there is a function $t_f : \mathbf{Z}_+ \to \mathbf{Z}_+$, where $\mathbf{Z}_+$ is the set of all positive integers, that satisfies

$\forall n \in \mathbf{Z}_+ \forall x \in \mathbf{Z}$
$((1 \leq x \leq n \Rightarrow \Phi_A^{t_f(n)}(c_s{}^{(n)})(x) = f) \wedge ((x < 1 \vee x > n) \Rightarrow \Phi_A^{t_f(n)}(c_s{}^{(n)})(x) = \#))$,
and
$\forall n \in \mathbf{Z}_+ \forall i \in \mathbf{Z}\ \forall x \in \mathbf{Z}(0 \leq i < t_f(n) \Rightarrow (\Phi_A^i(c_s^{(n)})(x) \neq f))$.

□

If A satisfies the above condition, we say A is a solution to the FSSP under the synchronizing condition SSC. The configuration $c_s^{(n)}$ is called an *initial configuration* for the FSSP, where cells $c_s^{(n)}(1), c_s^{(n)}(2), \cdots, c_s^{(n)}(n)$ should be synchronized. The function $t_f(n)$ is called a *firing time function* which gives the steps to synchronize these n cells. Further, $c_f^{(n)} = \Phi_A^{t_f(n)}(c_s^{(n)})$ is called a *firing configuration.*

2.4 Reversible and Conservative Cellular Automata

2.4.1 CA and Conservative Constraints

Reversibility and conservativeness are very important viewpoint when we use cellular automata for modeling physical phenomena.

A CA A is said to be *reversible* (or *injective*) iff its global function Φ_A is one-to-one.

Once Burks conjectured that any reversible cellular automata would not be computation-universal. But on the process of finding non-trivial RCAs, several techniques for constructing RCAs were developed [80] and Toffoli finally showed that an arbitrary d-dimensional *irreversible* CA can be embedded in a d+1-dimensional RCA, thus RCA can be computation-universal [79]. After this study, Margolus showed that his simple revesible BBMCA can directly embed the BBM (thus computation-universal) [47].

In general, the reversibility is defined as a property of global function of a CA, and thus it is very difficult to determine the reversibility from its local function. So it is also very difficult to give transition rules for a non-trivial RCA. In one-dimensional CAs, there is an algorithm to determine their reversibility from the transition rules [4], but in two-dimensional cases, their reversibility is proved to be undecidable [40, 41]. Hence it is hard to find a general method to construct reversible CAs.

In this paper, we use a partitioning technique to construct RCAs. In the following sections, we briefly describe computation-universal RCAs based on this technique.

2.4.2 Margolus Neighborhood

Conservative Logic and BBMCA

As mentioned in section 2.2, computation-universality of two- or higher-dimensional CAs can be proved by embedding universal logic elements. But on RCA, erasing informations is inhibited, and such irreversible logic gates can not be embedded directly. So when Margolus proposed his two-dimensional computation-universal RCA (BBMCA) [47], he realized its universality by embedding Fredkin and Toffoli's Billiard Ball Model (BBM) [21]. The BBM is a physical model of the conservative logic in which logical operations are performed by elastic collisions of balls. They showed that a 3-input 3-output reversible and bit-conserving Fredkin gate (F-gate) can be embedded in their BBM, and combining F-gates and unit delays, any logic circuits can be constructed on the BBM. On the BBMCA and Morita and Ueno's 16-state RPCA models, the BBM is simulated for showing their computation-universality. First we make a brief description about conservative logic and BBM.

A Fredkin gate (F-gate) is a basic element in the theory of Conservative Logic proposed by Fredkin and Toffoli [21] (This gate was first introduced by Petri, but the relation with conservative logic was first discussed by Fredkin and Toffoli). It is a reversible and bit-conserving logic gate and simply switches its two inputs 'p' and 'q' by the input 'c' (Fig. 2.6). They showed that AND, OR, NOT, and fan-out gates can be constructed by an

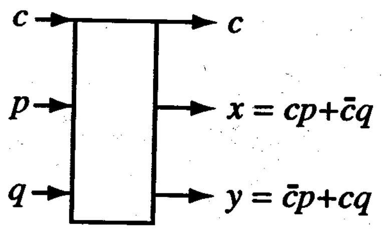


Figure 2.6: A Fredkin gate

F-gate (Fig. 2.7). Because of reversible and bit-conserving property, some constants must be put from the source lines and some garvage signals should be generated and come out along the sink lines depicted in Fig. 2.7. Using F-gates and unit delays, any m-input n-output function (Fig.2.8 (a)) can be embedded in a conservative logic circuit with constants and garvages depicted in Fig. 2.8 (b). Then they showed that such garvage signals can be "recycled" by the diagram in Fig. 2.9.

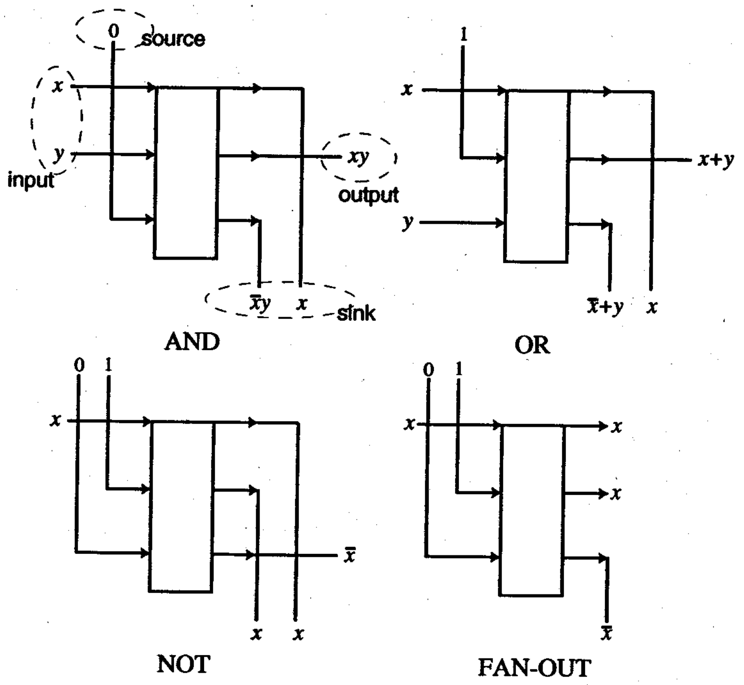


Figure 2.7: Construction of AND, OR, MOT, FAN-OUT gates by Fredkin gates

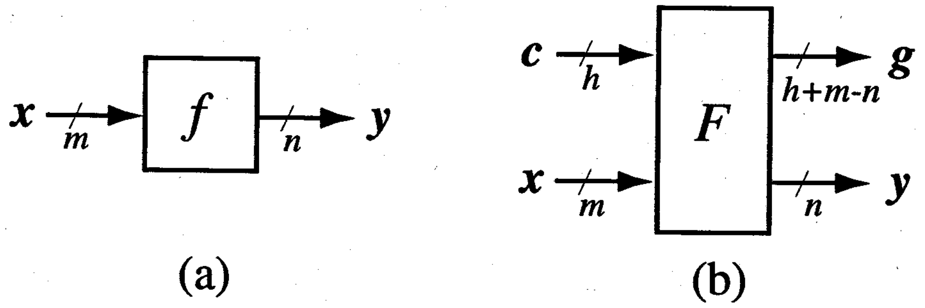


Figure 2.8: Any function can be embedded into a conservative logic circuit

They also introduced a switch gate (S-gate) and its inverse gate (Fig. 2.10). An S-gate is a 2-input, 3-output reversible and bit-conserving logic gate. An S-gate switches the input x by the control signal c. They showed that S-gates can be directly embedded in their BBM.

Fig. 2.11 shows the realization of an S-gate by balls. Using two S-gates and two inverse S-gates, an F-gate can be realized (Fig. 2.12). So any function can be computed without dissipation of balls as garbages. I.e., the BBM is a conservative model and it shows that it is possible to construct a computer that can computes with no energy dissipation in principle [11, 12].

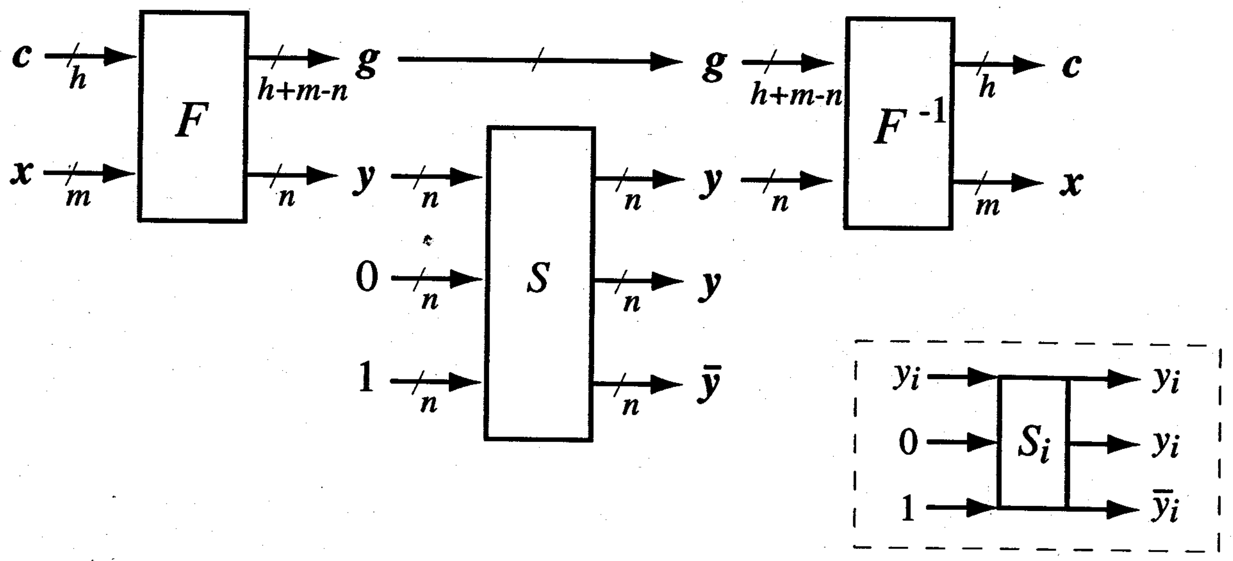


Figure 2.9: A garvageless computing of a function F

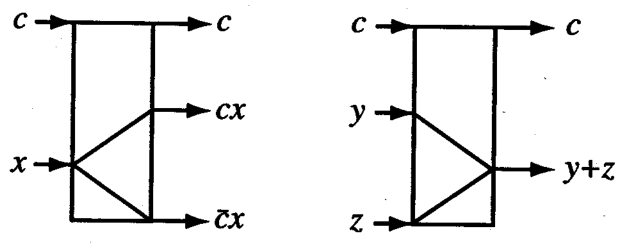


Figure 2.10: An S-gate and an inverse S-gate

Margolus introduced the BBMCA that can directly simulate the BBM. The BBMCA has alternating neighbor (Margolus neighbor) that varies depending on the parity of time depicted in Fig. 2.13. At the even time, grid of thick lines is used and thin grid is used at the odd time. Fig. 2.14 is a realization of a wall and reflection of "balls".

2.4.3 Partitioned CA (PCA)

Definition of Partitioned CA

Although the BBMCA is a simple RCA, it has a non-uniform neighbor. So Morita and Ueno proposed a different type of 4-neighbor 16-state computation-universal RCA [56]. They introduced partitioned cellular automata (PCA). PCA is regarded as the subclass of standard CA. Each cell is partitioned into the equal number of parts to the neighborhood size and the information stored in each part is sent to only one of the neighboring cells.

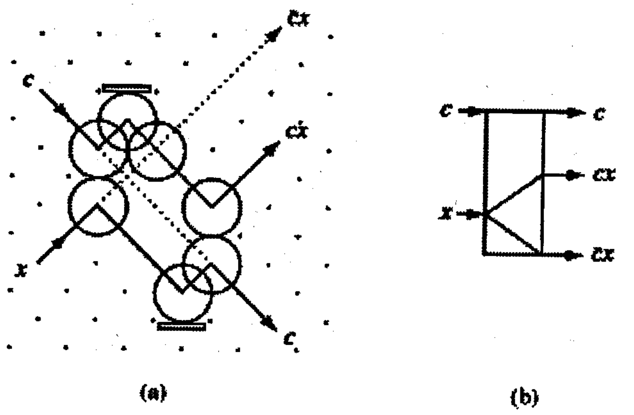


Figure 2.11: An S-gate by BBM

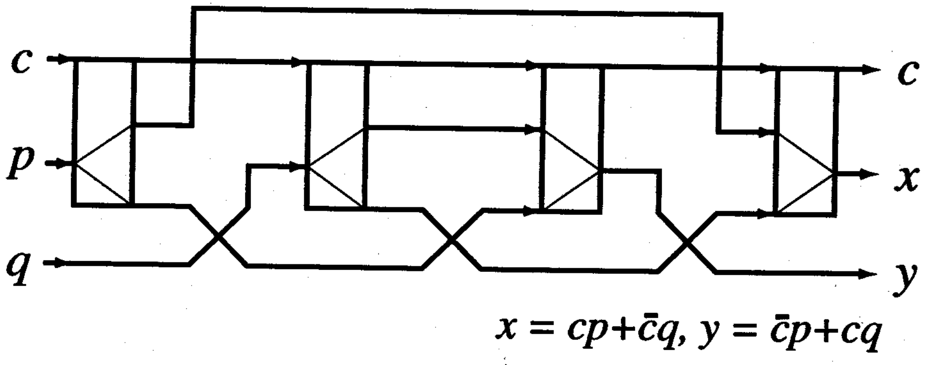


Figure 2.12: A realization of a Fredkin gate by S-gates and inverse S-gates

In PCA, injectivity of global function is equivalent to injectivity of local function, thus a PCA is reversible if its local function is injective [55]. An advantage of PCA is that it is easy to extend it into different number of dimensions and neighbors.

A deteministic one-dimensional three-neighbor PCA P is regarded as a special case of a normal one-dimensional CA, where each cell is partitioned into three parts L, C, and R. It is defined by

$$P = (\mathbf{Z}, (L, C, R), \varphi_P, (\#, \#, \#)),$$

where $\mathbf{Z}$ is the set of all integers, L, C, R is a non-empty finite sets of left, center and right internal states of each cell, $\varphi_P : R \times C \times L \to L \times C \times R$ is a mapping called a *local function*, and $(\#, \#, \#) \in L \times C \times R$ is a *quiescent state* which satisfies $\varphi_P(\#, \#, \#) = (\#, \#, \#)$.

A configuration over $L \times C \times R$ is a mapping $c : \mathbf{Z} \to L \times C \times R$. Let Conf($L \times C \times R$)

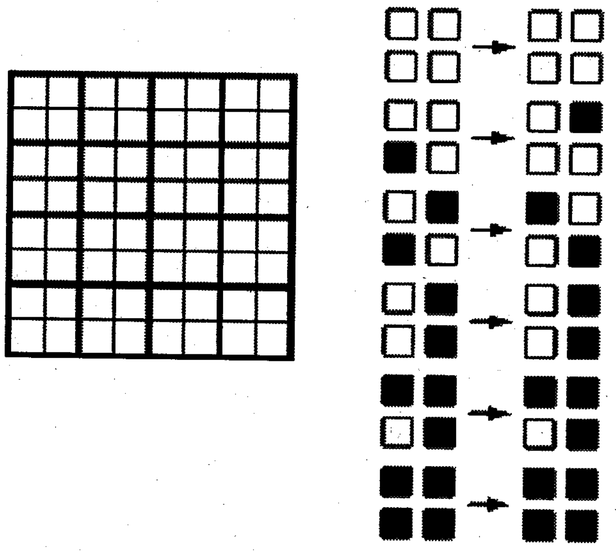


Figure 2.13: Margolous nighborhood and transition rules of BBMCA

denotes the set of all configurations over $L \times C \times R$.

$$\mathrm{Conf}(L \times C \times R) = \{c | c : \mathbf{Z} \to L \times C \times R\}$$

Global function

$$\Phi_P : \mathrm{Conf}(L \times C \times R) \to \mathrm{Conf}(L \times C \times R)$$

is also defined by

$$\Phi_P(c)(x) = \varphi_P(\mathrm{RIGHT}(c(x-1)), \mathrm{CENTER}(c(x)), \mathrm{LEFT}(c(x+1)))$$

where LEFT (CENTER, RIGHT, respectively) is the projection function which picks out the left (center, right) element of a triple in $L \times C \times R$. It has been proved that P is reversible iff φ_P is one-to-one [55]. Using PCA, we can construct a reversible CA with ease.

Definition of Two-Dimensional Partitioned CA

A deterministic two-dimensional partitioned cellular automaton (PCA) P is regarded as a special case of two-dimensional CA and defined by

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

where $\mathbf{Z}$ is the set of all integers, C, U, R, D, L are non-empty finite sets of center, up, right, down, left parts of each cell, $\varphi_P : C \times D \times L \times U \times R \to C \times U \times R \times D \times L$ is

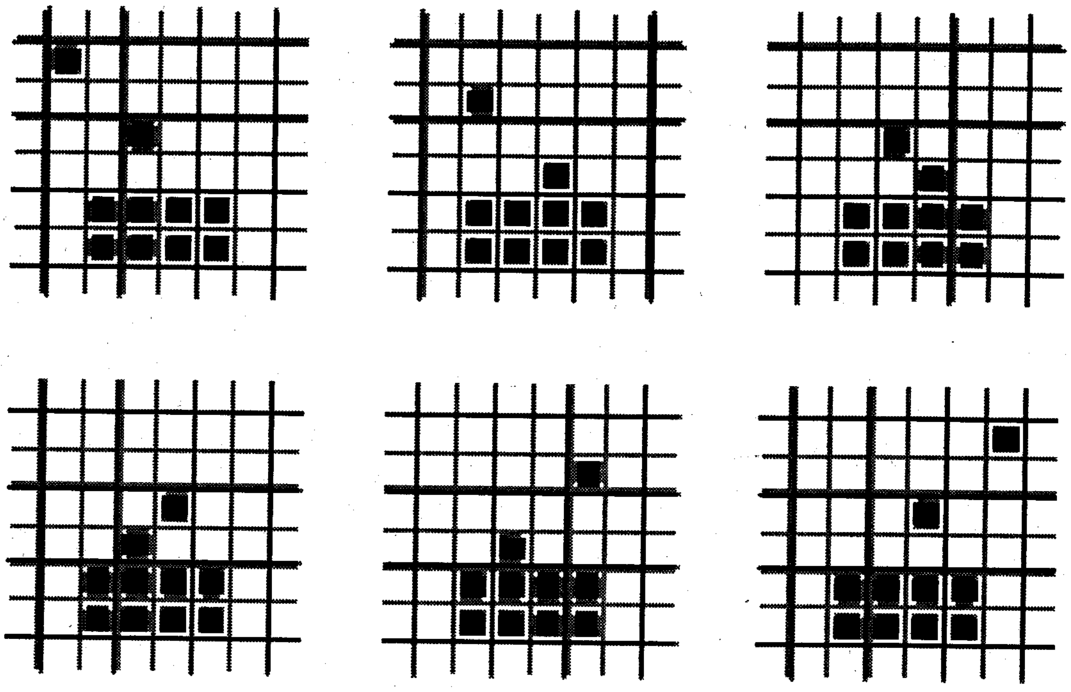


Figure 2.14: Ball reflection by BBMCA

a local function, and $(\#,\#,\#) \in C \times U \times R \times D \times L$ is a quiescent state which satisfies $\varphi_P(\#,\#,\#) = (\#,\#,\#)$.

A configuration over $C \times U \times R \times D \times L$ is a mapping $c : \mathbf{Z}^2 \to C \times U \times R \times D \times L$. Let Conf(C × U × R × D × L) denote the set of all cnfigurations over $C \times U \times R \times D \times L$.

$$\mathrm{Conf}(\mathrm{C} \times \mathrm{U} \times \mathrm{R} \times \mathrm{D} \times \mathrm{L}) = \{\mathrm{c} | \mathrm{c} : \mathbf{Z}^2 \to \mathrm{C} \times \mathrm{U} \times \mathrm{R} \times \mathrm{D} \times \mathrm{L}\}$$

Global function is also defined in the same way as in the one-dimensional case.

Computation-Universal Reversible PCA

Morita and Ueno's 16-state two-dimensional computation-universal model used a framework of 4-neighbor PCA, and Fig. 2.15 shows its domain and range of the local function. They proposed two models and Fig. 2.16 is the local function of one of their models. Their RPCA has following properties.

(i) Bit-conserving: the numbers of state "1" parts (i.e., black parts) on both sides of the transition rules are the same.

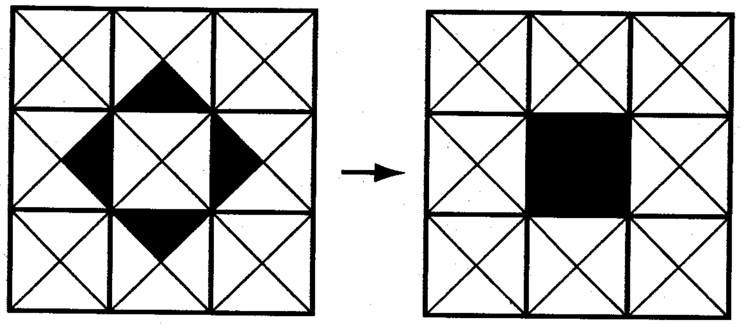


Figure 2.15: Domain and range of local function in a 2D 4-neighbor PCA

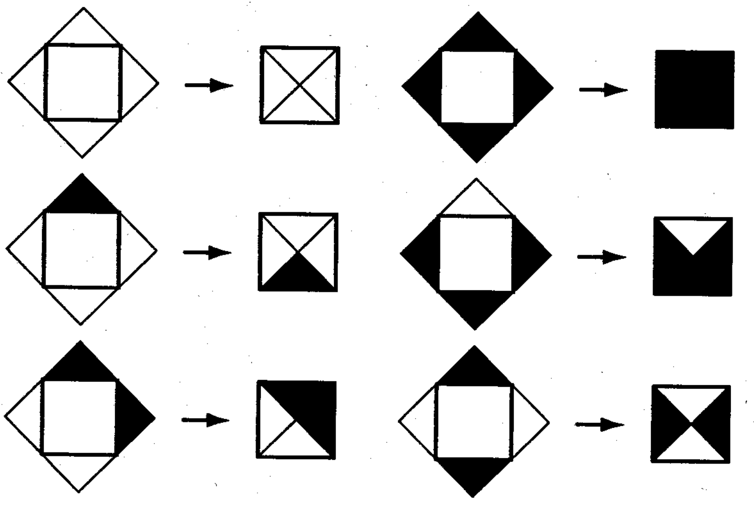


Figure 2.16: The local function of the 2D 16-state 4-neighbor RPCA

(ii) Reflection-symmetric

They proved its computation-universality by constructing S-gates, Inverse S-gates and F-gates on its cellular space. A "ball" is represented by two black parts depicted in 2.17. Fig. 2.18 shows a reflecting wall and Fig. 2.19 is a realization of an S-gate.

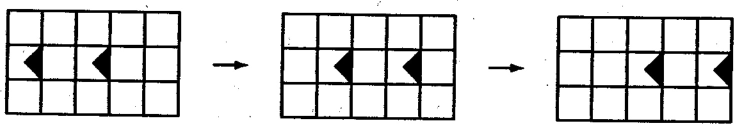


Figure 2.17: A ball in the 16-state 4-neighbor RPCA

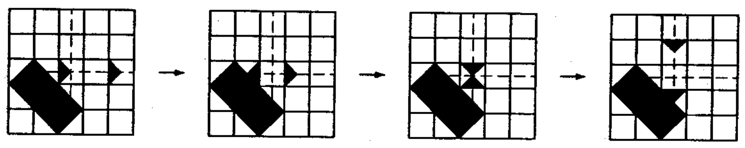


Figure 2.18: Ball reflection in the 16-state 4-neighbor RPCA

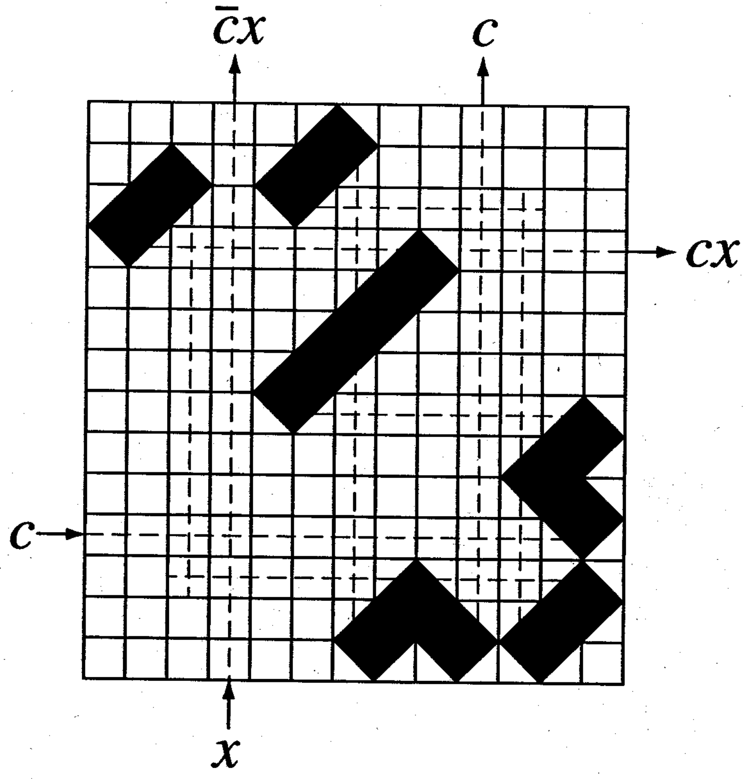


Figure 2.19: An S-gate in the 16-state 4-neighbor RPCA

Chapter 3

Computation-Universal Two-Dimensional Triangular Reversible CA

3.1 Definition of Triangular PCA

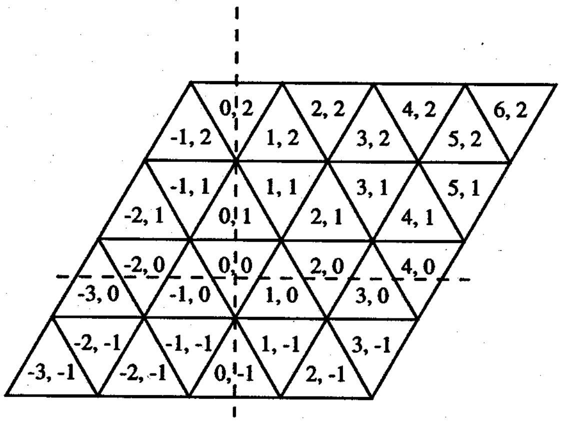
In this section, we show that the number of states of Morita and Ueno's models can be reduced. To decrease the number of states with preserving rotation-symmetry and bit-conservation, we used a triangular three-neighbor PCA, and thus 8-state RCA can be possible. This is the smallest state two-dimensional RCA under the condition of isotropic property in the framework of PCA.

First, we define a triangular neighbor PCA(TPCA). A TPCA has four neighbors if it has a center part. A three-neighbor TPCA is also available, if the center part is suppressed. We show 16-state and 8-state computation-universal reversible TPCA models, using four-neighbor and three-neighbor TPCAs respectively.

A four-neighbor TPCA (TPCA4) P_{T4} is defined as follows:

$P_{T4} = (\mathbf{Z}^2, (M, A, B, C), \varphi_{T4}, (\#, \#, \#, \#))$

- $\mathbf{Z}^2$ is the set of points where cells are placed ($\mathbf{Z}$ is the set of all integers).
- M,A,B,C are non-empty finite sets of four parts of a cell.
- $\varphi_{T4} : M \times A \times B \times C \rightarrow M \times A \times B \times C$ is a local function.
- $(\#, \#, \#, \#) \in M \times A \times B \times C$ is a quiescent state.



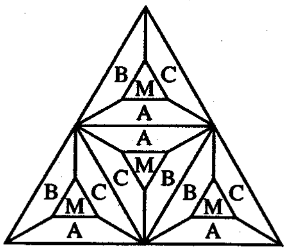


Figure 3.1: Index of 4-neighbor triangular RPCA

A configuration over the state set $Q = M \times A \times B \times C$ is a mapping $\alpha : \mathbf{Z}^2 \to Q$. Let Conf$(Q)$ denotes the set of all configurations over Q. Let $pro_x : Q \to X$ is a projection function ($X \in \{M, A, B, C\}$). The global function Φ_{T4} : Conf$(Q) \to$ Conf(Q) of P is defined as follows.

$\forall(x,y) \in \mathbf{Z}^2$, if $x+y$ is even, then
$\Phi_{T4}(\alpha)(x,y) = \varphi_{T4}(pro_M(\alpha(x,y)), pro_A(\alpha(x,y+1)), pro_B(\alpha(x+1,y)), pro_C(\alpha(x-1,y)))$
if $x+y$ is odd, then
$\Phi_{T4}(\alpha)(x,y) = \varphi_{T4}(pro_M(\alpha(x,y)), pro_A(\alpha(x,y-1)), pro_B(\alpha(x+1,y)), pro_C(\alpha(x-1,y)))$

We say P_{T4} is globally reversible iff Φ_{T4} is one-to-one, and locally reversible iff φ_{T4} is one-to-one.

P is called a *rotation-symmetric* TPCA iff (i) and (ii) hold.
(i) $A = B = C$
(ii) $\forall(m,a,b,c),(m,a',b',c') \in Q$
if $\varphi_{T4}(m,a,b,c) = (m',a',b',c')$ then $\varphi_{T4}(m,b,c,a) = (m',b',c',a')$

A rotation symmetric TPCA is called *reflection symmetric* iff (iii) holds.
(iii) $\forall(m,a,b,c),(m',a',b',c') \in Q$
if $\varphi_{T4}(m,a,b,c) = (m',a',b',c')$ then $\varphi_{T4}(m,a,c,b) = (m',a',c',b')$

Fig. 3.2 shows domain and range of the local function of P_{T4}. A three-neighbor TPCA P_{T3} can be defined by suppressing the center part M, and is denoted by $P_{T3} = (\mathbf{Z}^2, (A,B,C), \varphi_{T3}, (\#,\#,\#))$

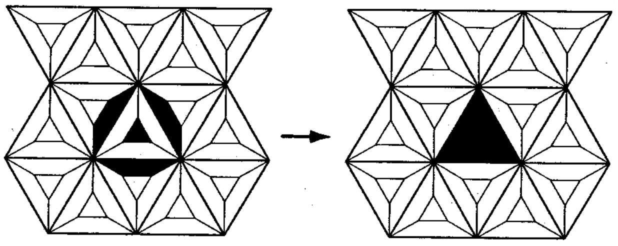


Figure 3.2: Domain and range of the local function in a 4-neighbor triangular PCA

3.2 16-state Triangular RPCA

A single cell of 16-state reversible TPCA4 can be regarded as a 3-input 3-output reversible logic gate with 1 bit internal state. Although an F-gate is also a 3-input 3-output gate, it cannot be possible to realize under the rotation-symmetric condition. But an S-gate and an inverse S-gate can be constructed under the rotation-symmetric condition. So the local rules of our model was chosen so that an S-gate (inverse S-gate) can be implemented by a single cell.

The local function of our proposed model is depicted in Fig. 3.3. Each part has 2-state and thus it is a 16-state reversible bit-conserving and rotation-symmetric one. Using this rule, an S-gate and an inverse S-gate can be embedded into a cell with the input/output relations depicted in Fig. 3.4. The rules used to realize the functions of S-gate and inverse S-gate are the rules where center parts are 0 (i.e., white).

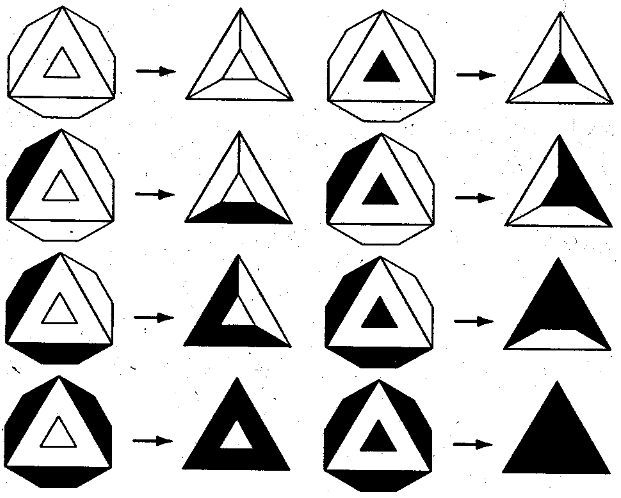


Figure 3.3: The local function of the 4-neighbor triangular RPCA

On the proof of computation-universality of the game of Life, gliders were used to

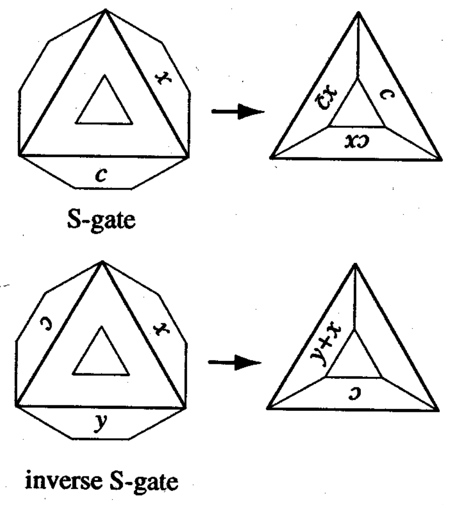


Figure 3.4: Construction of an S-gate and an inverse S-gate in the 4-neighbor triangular RPCA

encode signals. Similarly, the BBMCA and 16-state RPCA models used "ball"s. They propagate in a quiescent cellular space as signal carriers. But on TPCA, simple signals propagating in a quiescent cellular space cannot be constructed. To realize computation-universality, "wiring elements" i.e., data transmission wire, a delay element and a crossing element should also be given.

A data transmission wire is shown in Fig.3.5. Sequences of cells of this wire has alternating center part. A propagating signal turns right and left alternately, and thus it goes along the wire.

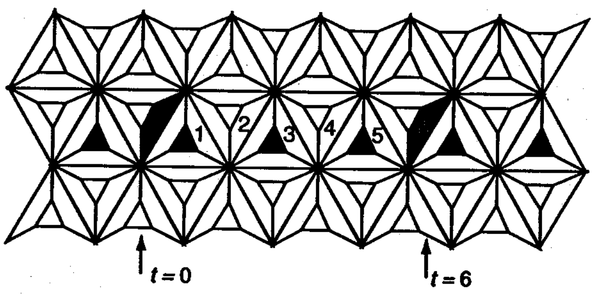


Figure 3.5: Data transmission wire in the 4-neighbor triangular RPCA

A delay element is shown in Fig. 3.6. This configuration act as a 2-step delay element.

The transmission wire depicted in Fig. 3.7 is an example of changing its row by two lines. Signals propagating on this wire has 2-step delay. Thus if the straight wire contains

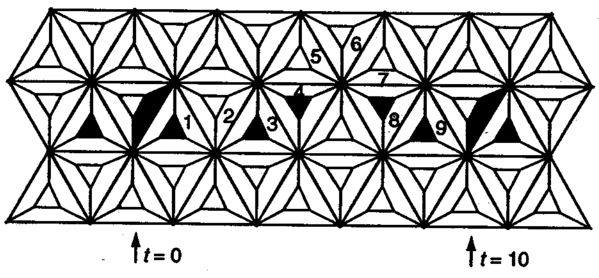


Figure 3.6: A delay element in the 4-neighbor triangular RPCA

a delay element as in Fig.3.6, both signals can be synchronized.

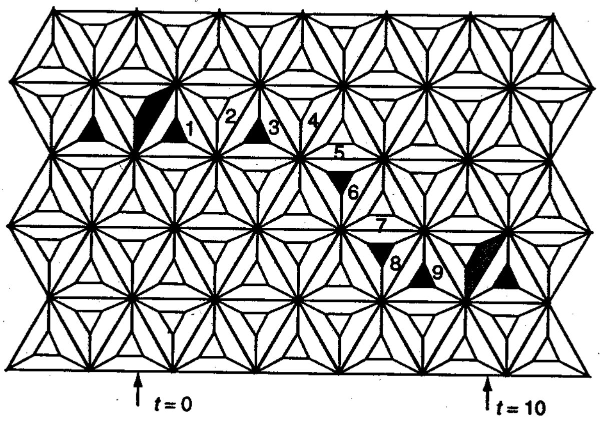


Figure 3.7: Changing signal orbit in the 4-neighbor triangular RPCA

Since this model uses wires for data transmission, we must give a data crossing mechanism. Fig. 3.8 shows a configuration for crossing wires. A cell of crossing point has black center part. Using these wiring elements and delays, it is possible to introduce signals to S-gates and inverse S-gates. An S-gate with synchronizing mechanism for input/output signals is shown in Fig. 3.9.

Combining circuit elements mentioned above, it is possible to construct an F-gate by the diagram of Fig.2.12, and thus any complex circuits can be assembled.

3.3 8-state Triangular RPCA

To decrease the number of states from Morita and Ueno's models with preserving bit-conservation and rotation-symmetry, we have to remove the center part from 16-state

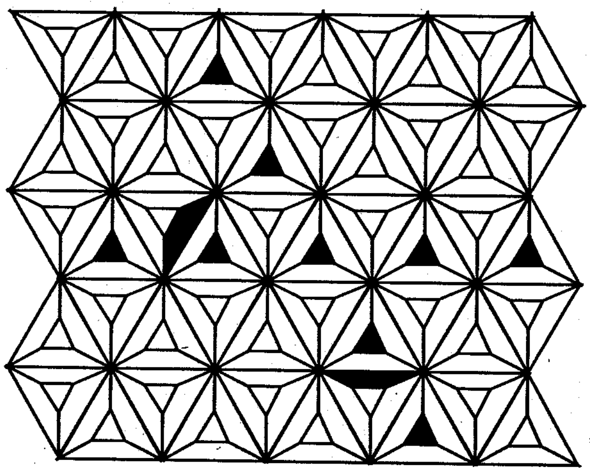


Figure 3.8: Crossing wire in the 4-neighbor triangular RPCA

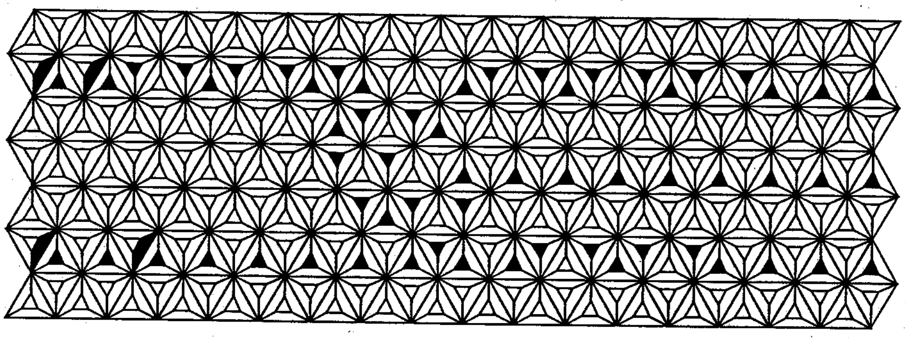


Figure 3.9: An S-gate with input/output lines in the 4-neighbor triangular RPCA

TPCA mentioned in the previous section. If it is possible, an 8-state computation-universal RPCA can be constractible. Fig.3.10 shows domain and range of a local function. This is the smallest state two-dimensional PCA under the rotation-symmetric condition.

There are nine bit-conserving and rotation-symmetric local functions by choosing four rules out of eight rules depicted in Fig.3.11 . And there are five different local functions if we exclude symmetric cases.

3.3.1 A Computation-Universal 8-state Model

We show that the RPCA where local function is given by Fig.3.12 is computation-universal.

First, we construct signal transmission wires.

In this model, it is difficult to construct simple patterns propagating in a quiescent

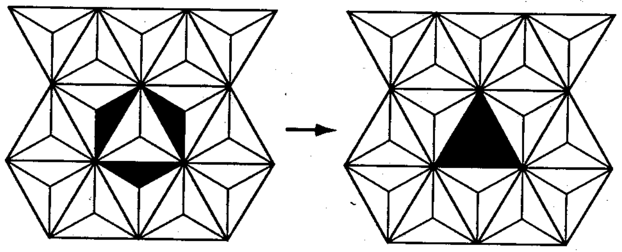


Figure 3.10: Domain and range of a local function in a 3-neighbor triangular PCA

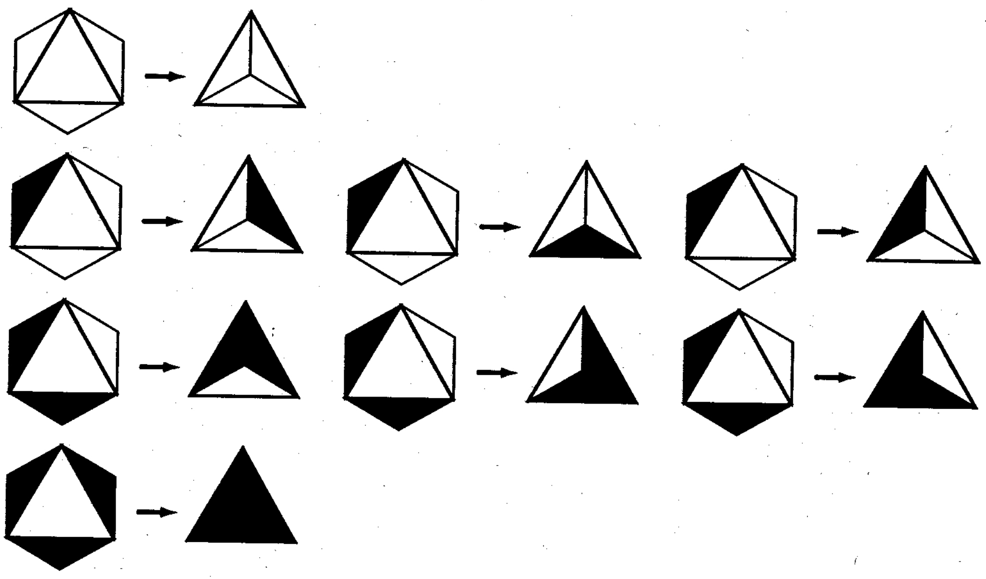


Figure 3.11: Rotation-symmetric transition rules in a 3-neighbor triangular PCA

cellular space. But there are two simple stable blocks shown in Fig.3.13 , and combining (b)-type blocks, it is possible to construct signal transmission wires (Fig.3.14). A signal carrier is shown in gray color. The gray signal takes 4 steps (regarded as 1 cycle) to move to the next dent (the part shown by the number 4).

A 4 steps (1 cycle) delay element is shown in Fig. 3.15 (a). This wire has a cave and signals take 4 steps in traveling through it. This delay element can be used as a synchronization element for changing columns/rows of transmission wires.

The transmission wire depicted in Fig. 3.15 (b) changes its row by two lines. The wire contains one right turning and one left turning, so signals propagating on the wire have strictly 4 step (1 cycle) delays. Thus if the straight wire contains a delay element (Fig. 3.15 (a)), both signals can be synchronized.

When a circuit contains feedback loop like Fig. 3.16 , the transmission wire turns

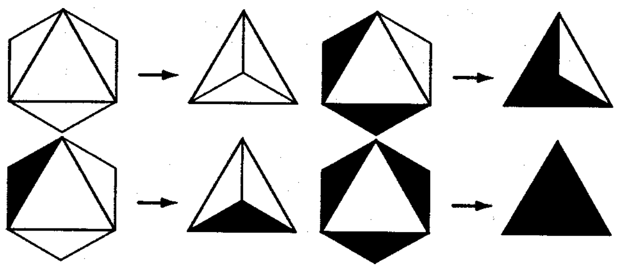


Figure 3.12: The local function of 3-neighbor triangular RPCA

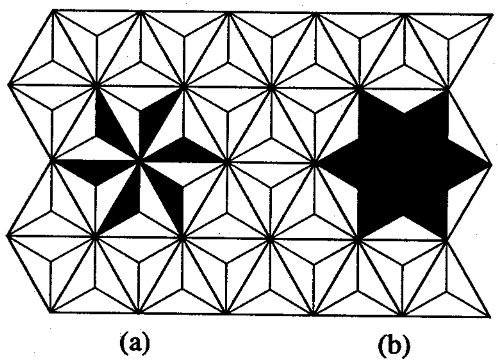


Figure 3.13: Stable blocks

right/left six times, and thus output signal phase of the feedback wire differs from that of input signal by ±2 steps(±0.5 cycles). Delay element of ±2 steps can be constructed by the crossing of two signals. Fig. 3.17(a) is a −2 steps phase shifter. The stable star shaped block above the horizontal wire has a "fin", and it turns around the 6 projections of the block in 30 steps. When this fin crosses the arriving signal along the wire, it advances the phase of the signal 2 steps. So this −2 steps phase shifter accepts signals every 30 steps. And combining a delay element, +2 steps shifter can be also available (Fig. 3.17(b)).

This model uses connected stable blocks for transmission wires and we need a special structure to realize crossing wires. Fig. 3.18 shows a module for crossing wires. Rotating "fin" switches signals from two input wires to each output wires. This crossing element accepts signals every 30 steps.

We showed that it is possible to realize signal transmission wires, delay elements, phase shifters and crossing wires. Next we show the function of S-gates and inverse S-gates can be available. Though, an F-gate cannot be realized under rotation-symmetric condition, an S-gate or an inverse S-gate is embedded into a single cell.

Fig. 3.19 shows the input/output relations of an S-gate and an inverse S-gate. Fig.3.20

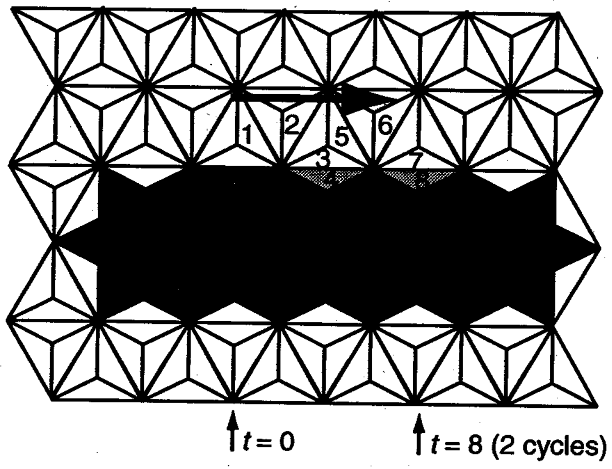


Figure 3.14: Data transmission wire

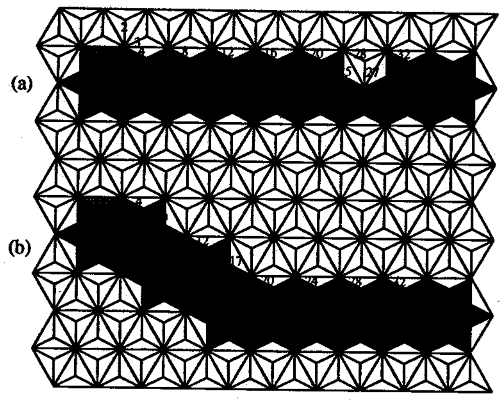


Figure 3.15: A delay element

shows an S-gate with synchronizing phase shifters. This element takes 40 steps (10 cycles) to get output signals, and inputs must be given every 30 steps.

Although all input/output signals in Fig.3.20 are synchronized, it does not completely fit the diagram of the S-gate in Fig.2.10. Because the output terminal of signal "c" is placed in the opposite side of this element. By improving this we show a complete pattern of an S-gate in Fig.3.21. In this pattern, an output signal "c" travells along a feedback wire and a crossing element, Thus it has a long delay against other output signals. So the right part of this patters are used to synchronize all the output signals. This element takes 140 steps to get output signals, and inputs must be given every 30 steps.

Because an S-gate and an inverse S-gate are symmetric, an inverse S-gate can be constructed by reflecting a pattern of the S-gate with simple modifications of rotating fins. It is shown in Fig.3.22.

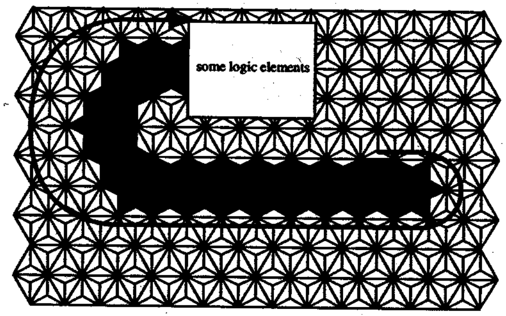


Figure 3.16: A feedback wire

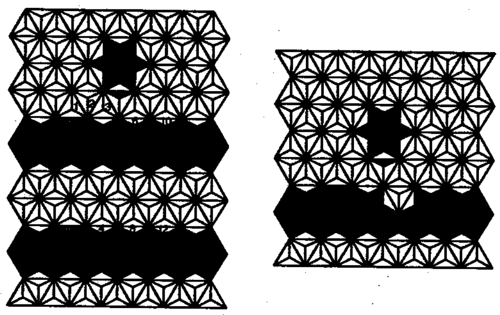


Figure 3.17: Phase shifters (a) −2, (b) +2

Combining the circuit elements mentioned above, an F-gate is constructed by using the diagram of Fig.2.12 (Fig. 3.23). It takes 704 steps (176 cycles) to simulate an F-gate.

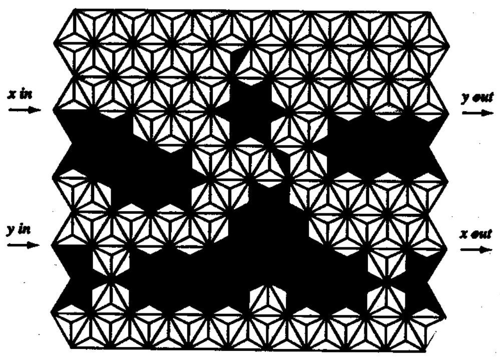


Figure 3.18: Crossing wire in the 3-neighbor triangular RPCA

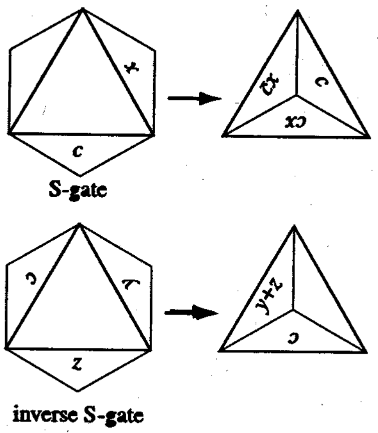


Figure 3.19: Construction of S-gate and inverse S-gate

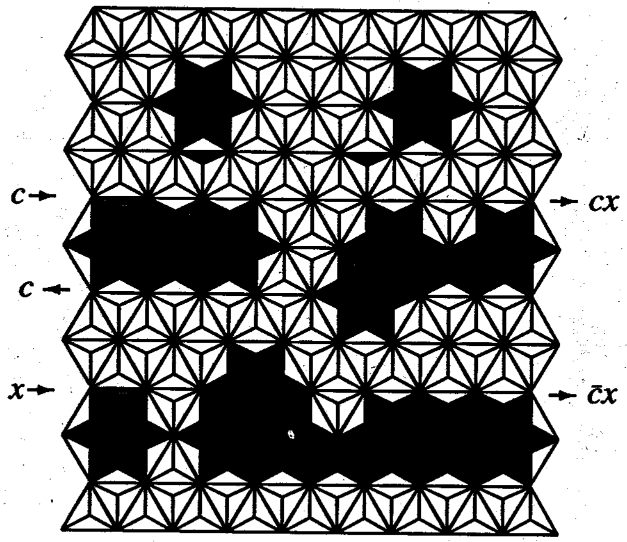


Figure 3.20: An S-gate with phase shifter

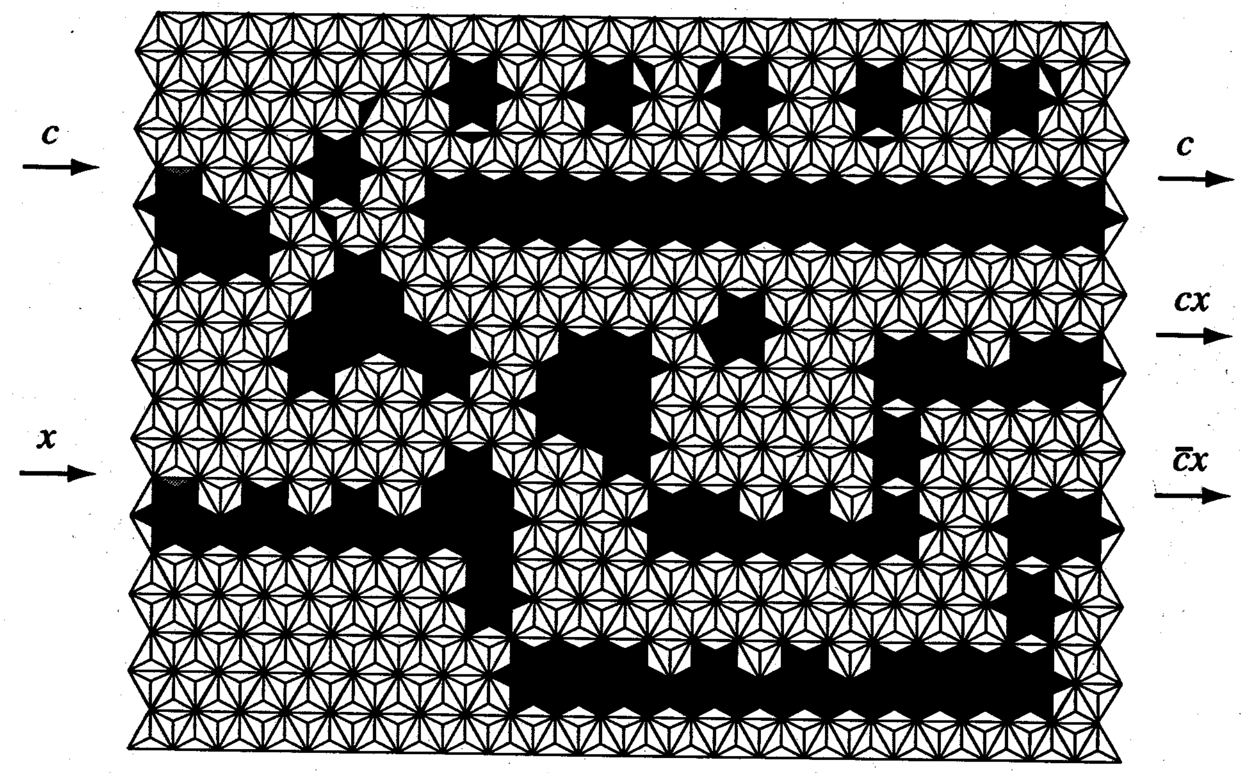


Figure 3.21: An S-gate with complete wiring

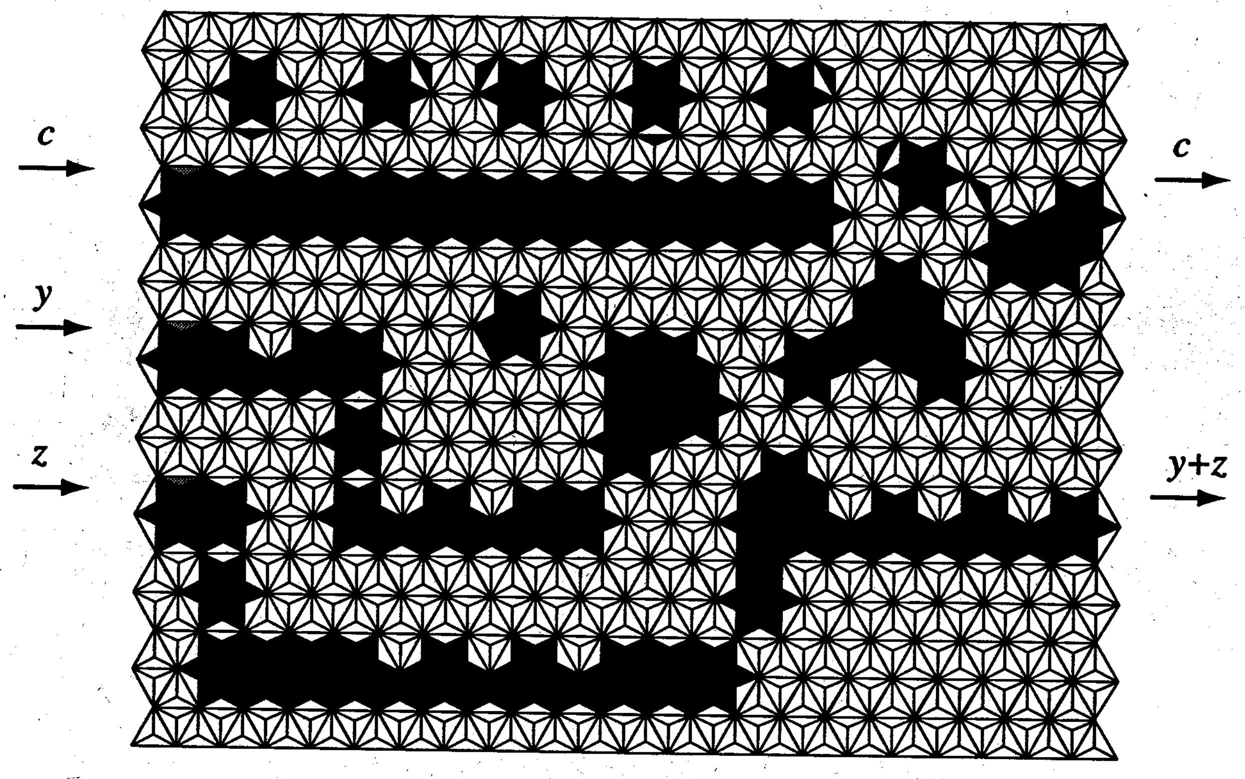


Figure 3.22: An inverse S-gate with complete wiring

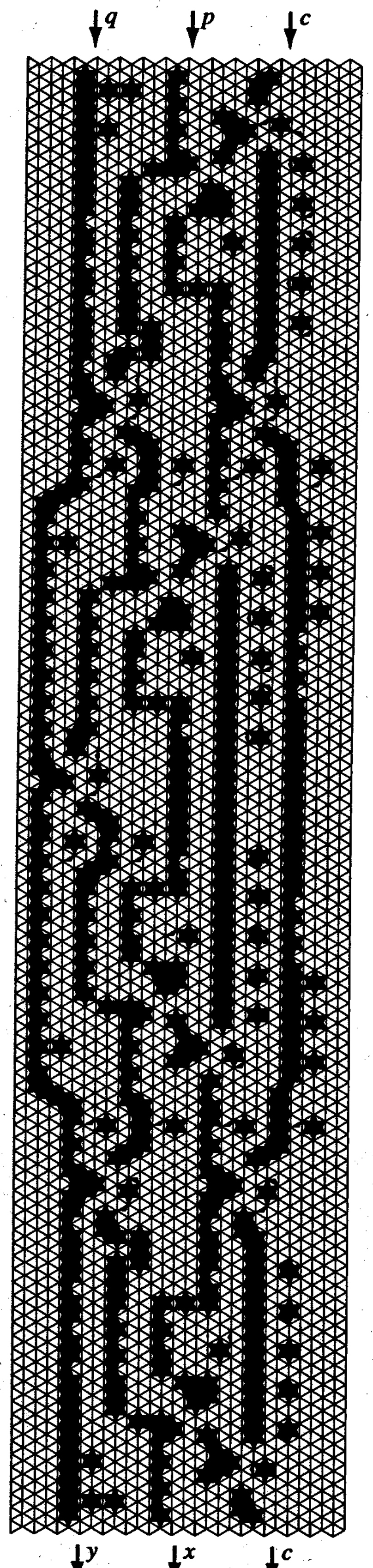


Figure 3.23: A configuration of an F-gate

Chapter 4

Synchronization on Reversible and Conservative Cellular Automata

4.1 Reversible Solutions for FSSP

In the previous chapter, we showed that it is possible to construct an 8-state computation-universal model on a reversible CA. But constraint of reversibility and conservativeness affects severely on the synchronizing activity, and it is the main difficulty of constructing reversible PCAs. So in this chapter, we try to construct several solutions to traditional Firing Squad Synchronization Problem (FSSP). But as we show in the next section, standard solutions where only one firing state is used cannot be constructed in a reversible cellular space. So we define a special weaker type of synchronizing condition.

Even if the weaker condition is introduced, signal generations are restricted. At first in section 4.1.2, we show a $3n$-time (Minsky type) reversible solution is possible.

In the following sections, we disscuss faster solutions to the FSSP in RCA. But Waksman-Balzar-Mazoyer type solutions are very difficult to embed into a reversible cellular space mainly because their methods of signal generations. Although both methods use recursive algorithms to synchronize, Minsky solution uses only constant kinds of signals (velocity 1 and 1/3) at every recursive stages and no other signals are generated. So it is fairly easy to cope with. Yet Waksman-Balzar-Mazoyer solutions use a recursive signal generation at each recursive stages, i.e., the number of signals with different velocity varies by the size of cells, and it is almost impossible to collect such signals. In RCA, un-collected signals are usually spreaded as garbages.

So we put a lower limit to the velocity of signals (hence a finite kinds of signals are used), and show 2^k-divided solution can be embedded into RCA to achieve faster synchronizing solutions. We assigned states and rules for the case of $k = 2, 3$. Although $(2+\varepsilon)n$-time solution could be possible, it is impossible to achieve an optimal time solution, or even $(2n+c)$-time one by this strategy.

But recently, Umeo [83] and Mazoer [50] reconstructed Goto's original optimal time solution. Goto's solution is very complex yet quite "traditional" and it become possible to construct an optimal time based reversible solution. We show our solution in section 4.1.4. Due to several reasons, our solution is not time optimal. It takes $2n+2$ steps to synchronize.

4.1.1 FSSP Conditions for Reversible Solutions

The synchronizing condition SSC in section 2.3.4 requires all n cells to become single firing state f on firing configuration. We can prove, in reversible CA, this type of solutions do not exist. It is proved using the next result by Richardson [71].

Proposition 4.1.1 [71] Let $A = (\mathbf{Z}, Q, (-1,0,1), \varphi_A, \#)$ be any CA, and Φ_A be the global function of A. If Φ_A is an injection, then there is a CA $B = (\mathbf{Z}, Q, N, \varphi_B, \#)$ whose global function is $\Phi_B = {\Phi_A}^{-1}$.

Theorem 4.1.1 There is no one-dimensional three-neighbor reversible CA which satisfies the synchronizing condition SSC.

Proof. Suppose there is a one-dimensional reversible CA $A = (\mathbf{Z}, Q, (-1,0,1), \varphi_A, \#)$ which satisfies SSC, and let $t_f(n)$ be the firing time function of A. By Proposition 4.1.1, there is a CA $B = (\mathbf{Z}, Q, N, \varphi_B, \#)$ whose global function is $\Phi_B = {\Phi_A}^{-1}$. Without loss of generality, we can assume $N = (m_1, m_2, \cdots, m_p) \in \mathbf{Z}^p$ and $|m_1| \leq |m_2| \leq \cdots \leq |m_p|$.

Assume, at time $t = 0$, B is in the configuration $c_f^{(n)} (= \Phi_A^{t_f(n)}(c_s^{(n)}))$. We consider the transition process from $c_f^{(n)}$ to $c_s^{(n)}$ on B, which is the reverse process from $c_s^{(n)}$ to $c_f^{(n)}$ on A. At time $t = 1$, all the $n - 2|m_p|$ cells of B at the positions from $|m_p|+1$ to $n - |m_p|$ are in the same state $\varphi_B(f, \cdots, f)$, because $c_f^{(n)}(x) = f$ holds for $1 \leq x \leq n$ and the next state of each cell depends only on the present states of the cells within the distance $|m_p|$. That is,

$$|m_p| + 1 \leq x \leq n - |m_p| \quad \Rightarrow \quad \Phi_B(c_f^{(n)})(x) = f^{(1)},$$

where $f^{(1)} = \varphi_B(f, \cdots, f)$. Generally,

$$i|m_p| + 1 \leq x \leq n - i|m_p| \quad \Rightarrow \quad \Phi_B^i(c_f^{(n)})(x) = f^{(i)} \tag{4.1}$$

holds for $i = 0, 1, \cdots, \lfloor (n-1)/2|m_p| \rfloor$, where $f^{(0)} = f$ and $f^{(j)} = \varphi_B(f^{(j-1)}, \cdots, f^{(j-1)})$ ($j = 1, 2, \cdots$).

Let s be the number of states of B (i.e., $s = |Q|$). Then, apparently there are $i_1, i_2 \in \mathbf{Z}$ that satisfy $0 \leq i_1 < i_2 \leq s$ and $f^{(i_1)} = f^{(i_2)}$ for large enough n. We claim that there exits k $(1 \leq k \leq i_2)$ such that $f^{(0)} = f^{(k)}$. If $i_1 = 0$ it is done. So consider the case $i_1 > 0$. Suppose $n \geq 2i_2|m_p| + 3$. Then, from (4.1), the followings hold for all $i \in \{0, 1, 2, \cdots, i_2\}$.

$$\begin{aligned}
&\Phi_B^i(c_f^{(n)})(\lceil n/2 \rceil - 1) = f^{(i)}, \\
&\Phi_B^i(c_f^{(n)})(\lceil n/2 \rceil) = f^{(i)}, \\
&\Phi_B^i(c_f^{(n)})(\lceil n/2 \rceil + 1) = f^{(i)}.
\end{aligned}$$

Further, since A is a three-neighbor CA such that $\Phi_B = {\Phi_A}^{-1}$,

$$\Phi_B^{i-1}(c_f^{(n)})(\lceil n/2 \rceil) = \varphi_A(\Phi_B^i(c_f^{(n)})(\lceil n/2 \rceil - 1), \Phi_B^i(c_f^{(n)})(\lceil n/2 \rceil), \Phi_B^i(c_f^{(n)})(\lceil n/2 \rceil + 1))$$

for all $i \in \{1, 2, \cdots, i_2\}$. Therefore $f^{(i-1)} = \varphi_A(f^{(i)}, f^{(i)}, f^{(i)})$, especially, $f^{(i_1-1)} = \varphi_A(f^{(i_1)}, f^{(i_1)}, f^{(i_1)})$ and $f^{(i_2-1)} = \varphi_A(f^{(i_2)}, f^{(i_2)}, f^{(i_2)})$ hold. Thus $f^{(i_1-1)} = f^{(i_2-1)}$ is obtained. By repeating this, we can conclude $f^{(0)} = f^{(k)}$ for $k = i_2 - i_1$. Consequently, the firing state f appears at time $t = k$ on B, or equivalently, appears at time $t = t_f(n) - k$ on A. Since $t_f(n) \geq 2n - 2$ and $0 < k \leq i_2 \leq s$, the relation $0 < t_f(n) - k < t_f(n)$ holds for large enough n. This contradicts the fact that A satisfies the synchronizing condition SSC. □

Synchronizing Condition for Reversible CA (RSC)

As shown above, In reversible CA, there is no solution with single firing state. So we define a set $F (\subset Q)$ of finite number of firing states, and regard that the cells synchronize if the state of each cell is in F at time $t_f(n)$. Moreover, we assume that not only the n cells but the other cells are allowed to change its internal states.

Let $A = (\mathbf{Z}, Q, (-1, 0, 1), \varphi_A, \#)$ be a reversible CA. A synchronizing condition RSC for reversible CA is as follows.

[RSC] There exist two distinct states $g, s \in Q - \{\#\}$ and a state set $F \subset Q - \{\#, g, s\}$ that satisfy the followings.

1. $\varphi_A(\#, s, s) = s$, $\varphi_A(s, s, s) = s$, and $\varphi_A(s, s, \#) = s$.

2. Let $c_s{}^{(n)}$ be a configuration defined by

$$c_s{}^{(n)}(x) = \begin{cases} g & x = 1 \\ s & x = 2, \ldots, n \\ \# & x \leq 0, x \geq n+1. \end{cases}$$

Then, there is a function $t_f : \mathbf{Z}_+ \rightarrow \mathbf{Z}_+$, that satisfies

$\forall n \in \mathbf{Z}_+ \forall x \in \mathbf{Z}$
$((1 \leq x \leq n \Rightarrow \Phi_A^{t_f(n)}(c_s{}^{(n)})(x) \in F) \wedge ((x < 1 \vee x > n) \Rightarrow \Phi_A^{t_f(n)}(c_s{}^{(n)})(x) \notin F))$,
and
$\forall n \in \mathbf{Z}_+ \forall i \in \mathbf{Z}\ \forall x \in \mathbf{Z}(0 \leq i < t_f(n) \Rightarrow (\Phi_A^i(c_s^{(n)})(x) \notin F))$.

4.1.2 $3n$-time Solutions

$3n$ Time Solution on Reversible PCA (99-state)

We present a 99-state solution where the numbers of elements of each state sets L, C, R are 3, 11, 3 respectively.

Solution on reversible PCA (99-state)

- $P = (\mathbf{Z}, L, C, R, \varphi_P, (\#, \#, \#))$, $C = \{\#, t, \vec{s}, \overleftarrow{s}, \vec{e}, \overleftarrow{e}, u\ \vec{w}, \overleftarrow{w}, v, f\}$, $L = R = \{\#, +, *\}$.

- General, soldier, and quiescent states are $(+, \vec{s}, *)$, $(\#, \overleftarrow{s}, \#)$, and $(\#, \#, \#)$ respectively. Initial configuration is:

$$c_s{}^{(n)}(x) = \begin{cases} (+, \vec{s}, *) & x = 1 \\ (\#, \overleftarrow{s}, \#) & x = 2, \ldots, n \\ (\#, \#, \#) & x \leq 0, x \geq n+1. \end{cases}$$

- Firing states set F is

$$F = \{(\#, f, +), (+, f, \#), (*, f, +), (+, f, *), (\#, f, \#), (+, f, +), (*, f, \#), (\#, f, *)\}.$$

- Transition table of local function φ_P is shown in Table 4.1.

Figure 4.1 shows transition of configurations ($n = 9$).

If we regard PCA P as a normal CA (denoted by A), then [RSC]1. becomes

$$\begin{aligned} \varphi_A((\#, \#, \#), (\#, \overleftarrow{s}, \#), (\#, \overleftarrow{s}, \#)) &= (\#, \overleftarrow{s}, \#) \\ \varphi_A((\#, \overleftarrow{s}, \#), (\#, \overleftarrow{s}, \#), (\#, \#, \#)) &= (\#, \overleftarrow{s}, \#) \\ \varphi_A((\#, \overleftarrow{s}, \#), (\#, \overleftarrow{s}, \#), (\#, \overleftarrow{s}, \#)) &= (\#, \overleftarrow{s}, \#). \end{aligned}$$

Since $\varphi_P(\#, \overset{\leftarrow}{s}, \#) = (\#, \overset{\leftarrow}{s}, \#)$ holds, the above condition [RSC]1. is satisfied. The reversiblity of P is concluded by the fact that φ_P is one-to-one.

This solution is based on Minsky's $3n$ time one. First, the leftmost cell (general) emits two kinds of signals of velocity 1 and 1/3 to the right. At time $3n/2$, they collide at the center of the n cells, and thus the problem is divided into two subproblems of synchronizing $\lfloor n/2 \rfloor$ cells. Then the center cell(s) becomes a new general, and emits these signals in both directions. Repeating this process, it is finally reduced to the problem of synchronizing single cell. Propagation and bouncing of signals can be easily performed reversively on a PCA. But, in order to make it completely reversible, the following information must be kept in the cells where the collisions of signals have occurred: (1) the direction of the velocity 1/3 signal, and (2) the phase of the collision of two signals (it depends on the parity of the length of the array). They are memorized by an "arrow" of the center part state and the state "+" of the left and right parts. They can be thought as "garbage" information.

From the construction of P, we can see the following facts. If the length n of the array is even, the $(n+2)/2$-th cell from the general becomes a new general for the two arrays of length $n/2$ at time $3n/2$ (hereafter the old general acts as a "wall" that bounces the signal of velocity 1). If the length n is odd, the $(n+1)/2$-th and the $(n+3)/2$-th cells from the general become new generals for the two arrays of length $(n-1)/2$ at time $3(n+1)/2$. By above, $t_f(n) = 3n$ is concluded (by a simple induction on n).

In general, simulating an irreversible CA by a reversible one, "garbages" spread over. But in this solution, all signals (including signals introduced to keep reversibility) are confined to the $n+2$ cells and do not spread over the entire cell space. We call it a *garbage preserving reversible* CA.

Table 4.1: Transition table of 99-state reversible PCA P ($\overleftarrow{s}$ and $\overleftarrow{w}$ are symmetric with $\overrightarrow{s}$ and $\overrightarrow{w}$.)

$\#$	$\#$	$*$	$+$
$\#$	$\#\#\#$	$\#\#*$	$\#\#+$
$*$	$*\#\#$		
$+$	$+\#\#$		

$\overrightarrow{s}$	$\#$	$*$	$+$
$\#$	$\# \overrightarrow{s} \#$	$* t +$	$* u +$
$*$		$+ \overrightarrow{e} +$	$\# \overrightarrow{w} +$
$+$		$\# \overrightarrow{e} +$	$+ \overrightarrow{w} +$

t	$\#$	$*$	$+$
$\#$	$\# t \#$	$* \overleftarrow{s} \#$	$+ t \#$
$*$	$\# \overrightarrow{s} *$		$* \overleftarrow{s} *$
$+$	$\# t +$	$* \overrightarrow{s} *$	$\# u \#$

u	$\#$	$*$	$+$
$\#$	$\# v \#$	$\# \overleftarrow{s} *$	$\# \overleftarrow{s} +$
$*$	$* \overrightarrow{s} \#$		
$+$	$+ \overrightarrow{s} \#$		

v	$\#$	$*$	$+$
$\#$	$* u *$	$\# v +$	$\# \overrightarrow{s} +$
$*$	$+ v \#$		
$+$	$+ \overleftarrow{s} \#$		

$\overrightarrow{w}$	$\#$	$*$	$+$
$\#$	$\# \overrightarrow{w} \#$	$\# \overrightarrow{w} *$	$\# f +$
$*$		$* \overrightarrow{w} *$	
$+$	$+ \overrightarrow{w} \#$	$+ \overrightarrow{w} *$	$* f +$

$\overrightarrow{e}$	$\#$	$*$	$+$
$\#$	$\# f \#$		
$*$			
$+$	$* f \#$		

$\overleftarrow{e}$	$\#$	$*$	$+$
$\#$	$+ f +$		$\# f *$
$*$			
$+$			

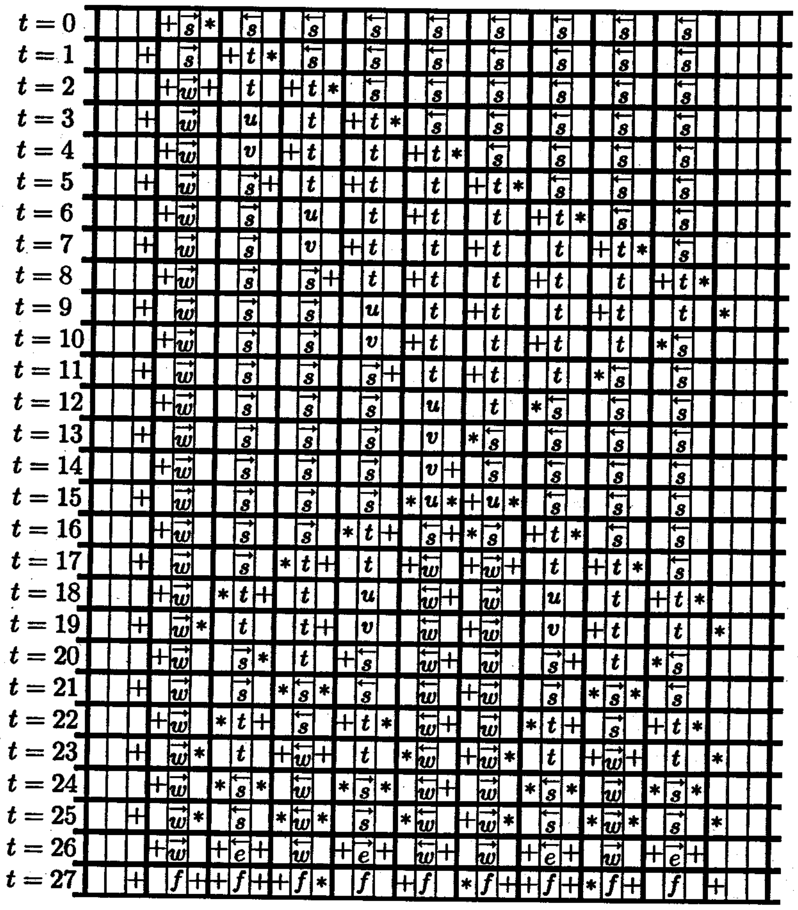


Figure 4.1: Synchronization of 9 cells by the 99-state reversible PCA P.

4.1.3 2^k-divided Solutions on Reversible PCA

2^k-divided Solutions

$3n$-time based solutions use the collision of two signals (1 and 1/3) and find the center cell and divide the problem into two sub problems. Waksman realized an optimal time ($2n - 2$ time) solution [84] using slower signals ($1/7, 1/15, 1/31, \cdots$) at a time, i.e., Waksman's solution generates slower signals recursivly. But in reversible CA, this approach seems to be very difficult to apply, because their restriction of generating/erasing signals. So cutting off infinite generation of such signals and using only k slower signals, 2^k-divided solutions can be realized in reversible spaces (k is any finite non-negative integer). We construct examples of this type of solutions in $k = 2, 3$ in this section, and thus we show $(2 + \varepsilon)n$-time solution can be possible ($\varepsilon > 0$).

Construction of a 2^2-divided Solution

Fig.4.2 is the outline of a 2^2-divided solution. $1/7$ signal is also used with 1 and $1/3$ signals and soldiers are divided into four parts after about $7n/4$ steps.

Generally, 2^k-divided solution takes $(2-1/2^k)n$ steps for the first division and the firing time is

$$t_f(n) = \frac{2^{k+1}-1}{2^k-1}n$$

So the firing time for a 2^2-divided solution is $7n/3$.

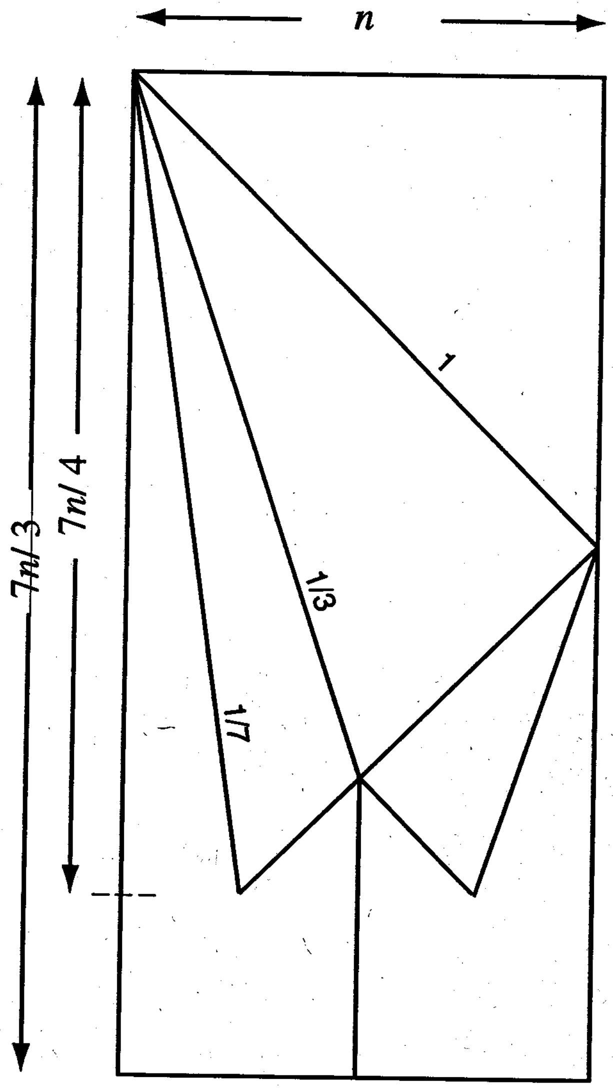


Figure 4.2: The outline of 2^2 divided solution

We have constructed an example of 2^2-divided solutions.

2^2-divided solution P_4

- $P_4 = (\mathbf{Z}, L_4, C_4, R_4, \varphi_{P_4}, (\#, \#, \#))$,
  $C_4 =$ { G1R, G1L, G2R, G2L, G3R, G3L, GA1R, GA1L, GA2R, GA2L, GA3R, GA3L, GB1R, GB1L, GB2R, GB2L, GB3R, GB3L, GC1R, GC1L, GC2R, GC2L, GC3R, GC3L, GD1R, GD1L, GD2R, GD2L, GD3R, GD3L, #, U1R, U1L, U2R, U2L, V1R, V1L, V2R, V2L, V3R, V3L, V4R, V4L, V5R, V5L, V6R, V6L, S, AR, AL, GA0L, GA0R, BR, BL, GB0R, GB0L, WR, WL, XR, XL, YR, YL, GC0R, GC0L, G0R, G0L, PR, PL, QR, QL, GD0R, GD0L, HR, HL, HAR, HAL, HBR, HBL, HCR, HCL, HDR, HDL, EUR, EUL, ESR, ESL, EHR, EHL, EHAR, EHAL, EHBR, EHBL, EHCR, EHCL, EHDR, EHDL, FUR, FUL, FSR, FSL, FEHR, FEHL, FEHAR, FEHAL, FEHBR, FEHBL, FEHCR, FEHCL, FEHDR, FEHDL, FHR, FHL, FHAR, FHAL, FHBR, FHBL, FHCR, FHCL, FHDR, FHDL, FG1R, FG1L, FGA1R, FGA1L, FGB1R, FGB1L, FGC1R, FGC1L, FGD1R, FGD1L, FGR, FGL, FGAR, FGAL, FGBR, FGBL, FGCR, FGCL, FGDR, FGDL },
  $L_4 = R_4 = \{--, //, ++, /, *, -, **, \#, +\}$,

- General is $(*, \mathrm{G0R}, *)$, soldier is $(\#, \mathrm{S}, \#)$, quiescent state is $(\#, \#, \#)$. Initial configuration is
$$c_s{}^{(n)}(x) = \begin{cases} (*, \mathrm{G0R}, *) & x = 0 \\ (\#, \mathrm{S}, \#) & x = 1, \ldots, n-1 \\ (\#, \#, \#) & x \leq 0, x \geq n. \end{cases}$$

- The number of elements of the Firing state set F_4 is 146.

- Local function is composed of 376 transition rules.

The number of states $n(L), n(C), n(R)$ are 9,140,9 respectively and thus 11340 states and because of the injectivity of its local function, P_4 is reversible.

Fig 4.3 shows its configuration for n=11.

Construction of 2^k-divided Solutions ($k = 3$)

In this section, we construct a 2^3-divided solution more systematically and show that 2^k-divided solutions for arbitrary ($k \leq 2$) can be constructed, i.e., we show $(2+\varepsilon)n$-time solution can be achieved.

Main problems are the following:

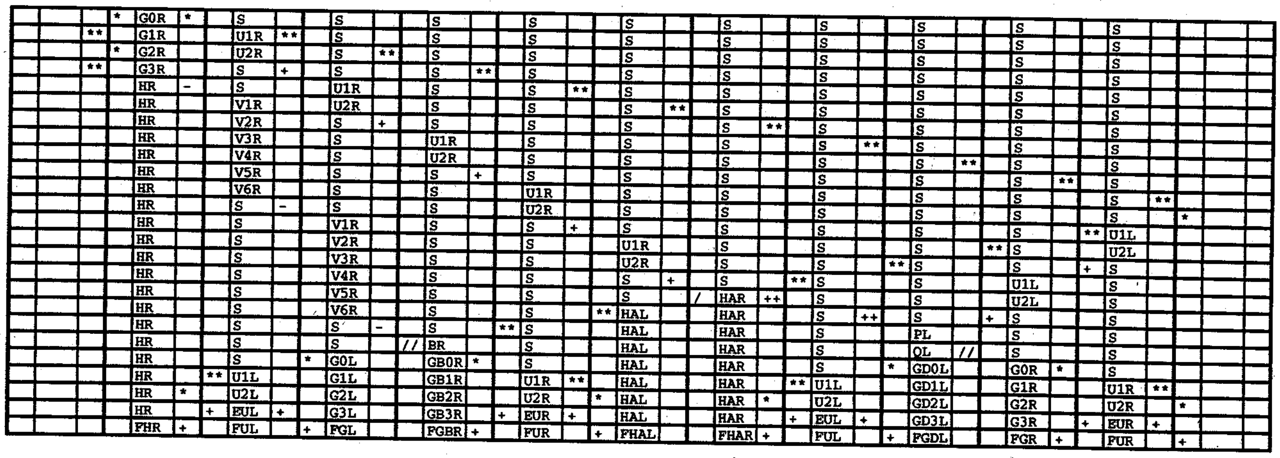


Figure 4.3: A configuration of 2^2-divided solution (n=11)

- Can all new nodes (new generals) be generated by collisions of signals?
- Can all new generals be placed at the center of cells?
- Can next four generals be generated at the same time?
- Can all signals be generated in reversible manner?

We will denote about these four points.

A. Signals make collisions

On a 2^3-divided solution, after generated b, c_0, d_0 nodes, four nodes $a_{i,j}$ are generated as new generals (Fig.4.4). All nodes except b are generated from a same node, and these nodes are created by the collision of two signals 1 and one of the opposite signal of $1/3, 1/7, 1/15, \cdots$.

On usual CA, two opposite signals must make a collision, but on PCA, these signals do not always make a collision. Fig. 4.5 shows the difference of phase between $1/3$ and -1 signals when these signals make a collision. Signals depicted in (1) make a collision, but in (2), two signals do not make a collision and signals are crossing. But on this 2^k-divided solution, one can set the initial phase of signals that all these signals should make collisions. Let the position of the first signal with velecity 1 from general at time t be $s_1(t)$. After reflected by the right wall, it exists on the L partition of a cell which position satisfies the number $t + s_1(t)$ is always even. And slow signals for a collision have velocity $1/m$ (m is odd). If their initial phase (i.e., the delay time for moving to the next cell) are even, the moving to the next cell occurs only on a cell which satisfies the sum of the time and the position of the cell is even. This is the same

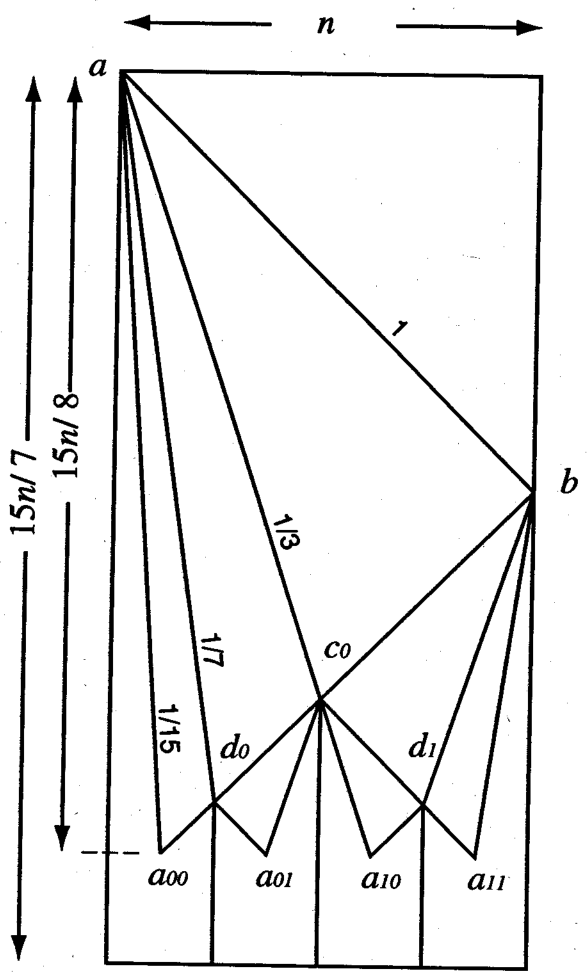


Figure 4.4: The outline of 2^3-divided solution

as the velocity 1 signal and two signals always make a collision and it never occur the crossing signals like Fig. 4.5 (2).

Same discussion can be possible in the odd case and a condition for signal carriers can be denoted as follows.

[A condition for state of signal carriers] : On time t, each cells (say position x) which contain carriers of signals of velocity $1, 1/3, 1/7, \cdots$ in its L or R partitions should satisfy that $t + x$ are all even or all odd.

To satisfy above condition, initial phase of each signals are set by table 4.2.

Using this initial phase, only (1) and (3) patterns in Fig. 4.5 occur on collisions by velocity 1/3 signals.

B. Divisions are made in equal length of four part

(1)

	s			s			s			s	
2	s			s			s			s	
	11R			s			s			s	
	12R			s			s		1	s	
	s	2		s		1	s			s	

(2)

	s			s			s			s	
2	s			s			s			s	
	11R			s			s		1	s	
	12R			s		1	s			s	
	s	2	1	s			s			s	

(3)

	s			s			s			s	
2	s			s			s		1	s	
	11R			s		1	s			s	
	12R		1	s			s			s	
*	*	*		s			s			s	

(4)

	s			s			s		1	s	
2	s			s		1	s			s	
	11R		1	s			s			s	
*	*	*		s			s			s	
*	*	*	*	*	*		s			s	

Figure 4.5: Phase of collision signals

Each nodes c, d and a new general a should be generated to divide cells into two parts of equal length. If the number of cells is odd, single center cell is assigned to a new node and if it is even, two cells are assigned and the number of the divided cells are kept to same number.

Fig. 4.6 is a part of configuration that new node c is generated. (1) is the case of $n = 8$, i.e., the number of the cells to be divide is 7 (odd) and state c is assigned as a new node for a center cell. On the other hand, if the number of the cells to be divide is even, symbol cL and cR are used to denote generated new nodes for two center cells (2a), (2b).

As depicted in (2a), if the node c generate signal "$c3$" to both directions at $t = 14$, both signals are synchronized correctly. But the right signal violate the condition of signal carriers. So "$c3+$" is generated one step before the correct time and "+" symbol remembers the difference of the time.

C. Can next four generals a be generated at a time?

As mentioned above, according to the condition for signal carriers, signals should preserve the difference from their true positions by themselves. A 2^3-divided solution has 8 cases according to n (table 4.3). The number (0 to 3) in table 4.3 denotes the difference of true position of

Table 4.2: Initial phase of signals

speed	initial phase
1	0
1/3	-2
1/7	-2
1/15	-8

generated nodes from collision point signals. If an item contains R or L, it shows that two cells are used for the new node and R and L are directions from the collision point which cell is another node. I.e. R (L) means that a right (left) neighbor cell of collision point becomes also a new node.

Table 4.3: Construction of each nodes

$n(mod8)$	c	d_0	d_1	a_0	a_1	a_2	a_3
0		0	0	0	0	0	0
1	R	0	1	0	0	1	1
2		0R	0L	0	1	1	0
3	R	0R	1L	0	1	2	1
4		0	0	0L	1L	1R	0R
5	R	0	1	1R	1L	2R	1R
6		0R	0L	1R	2L	1R	1L
7	R	0R	1L	1R	2L	3R	2L

To synchronize the generated time of four new general $a_{i,j}$, specific delay steps should be inserted according to table 4.3. Delay steps k for 2^3-divided solution is at most $k(=3)$.

Although at $t = 0$ general a generates signals only single direction, new generals of the other successive recursive stages generate signals to both directions. But both signals are symmetric and no new phase composition appears.

D. Reversibility

To achieve reversiblity, local function of the PCA should be injective. And to assure of this injectivity, at all collision points, directions and velocities of signals should be preserved after each node generations. It is possible to embeded such informations into the subcases of state of each nodes and we show the state table in case of general $a_{i,j}$ node on table 4.4.

(1) $n = 8$

		3			4			5			6	
9		s	2		s			s			s	
10		s			11R			s		1	s	
11		s			12R		1	s			s	
12		s		c3	c0	c3		s			11L	
13	c1	c11L			c0			c11R	c1		12L	
14		c12L			c0			c12R		d1	d01	d1
15	3	a010	3		c0		3	a020	3		d01	

(2a) $n = 9$

		3			4			5			6	
9		s	2		s			s			s	
10		s			11R			s			s	
11		s			12R			s			s	
12		s			s	2		s		1	s	
13		s			s		c3*	cR!			s	
14		s		c3	cL			cR	c3		s	
15	c1	c11L			cL			cR			c11R	c1

(2b) $n = 9$

		3			4			5			6	
9		s	2		s			s			s	
10		s			11R			s			s	
11		s			12R			s			s	
12		s			s	2		s		1	s	
13		s			s		c3*	cR	c3+		s	
14		s		c3	cL			cR			c11R	c1+
15	c1	c11L			cL			cR			c12R	

Figure 4.6: The difference of phase of collisions by the difference of parity

In the case of $n = 4, \cdots, 8(mod8)$, generated a nodes are always denoted aL, aR (see table 4.3), Reflecting signals between aL and aR preserve phase and directions of signal collision such as a state "42+" in fig. 4.7.

An Example of 2^3-divided Solution

To construct transition rules for 2^3-divided solution all cases should be assigned. Fig. 4.8 shows a configuration of $n = 18$. 2^3-divided solution has 64 subcases for generating the next generation of generals $a_{i,j}$.

4.1.4 Minimal Time Solutions and Reversibility

As mentioned in previous section, it is possible to construct $(2 + \varepsilon)n$-time solution. But the number of its state should be huge and it is not appropriate way to construct faster synchronizing solutions. Recently, completely different algorithm can be reconstructed based on Goto's algorithm by Mazoyer [50] and this leads to a construction of fairly small state minimal time so-

Table 4.4: Subcases of general $a_{i,j}$

n(mod 8)	a_0	a_1	a_2	a_3
0	$a000$	$a010$	$a020$	$a030$
1	$a100$		$a120$	$a130$
2	$a200$	$a210$		$a230$
3	$a300$		$a320$	$a330$
4	40+	41+	42+	43+
5	50+		52+	53+
6	60+	61+		63+
7	70+		72+	73+

		6			7			8			9	
20		c0	+		c12R			s	c1		12L	
21		c0			s	c2		s		d1	d01	d1
22		c0			s		42+	aR!			d01	
23		c0			aL+	42+		aR+			d01	
24		c0		3	a0L		42+	a0R	3		d01	
25	+	ec0	+		a1L	42+		a1R		+	ed01	+
26		fec0			fa1L		42+	f1R			fed01	

$n = 12$

Figure 4.7: Reservation of collision phase in generating general a

lutions. This method is based on an idea that construct sublines of length 2^i and to synchronize each regions using $3n$-time solutions.

In this section, we construct a reversible solution based on this algorithm.

Dividing into Two Regions

First, n cells are divided into two part of same length using signals s_1 (velocity 1) and s_3 (velocity 1/3) as follows (Fig.4.9). This is the same approach as $3n$-time solution.

- the case of n is odd[1,$\lfloor \frac{n}{2} \rfloor + 1$ (G_1)],[$\lfloor \frac{n}{2} \rfloor + 1$,n]
- the case of n is even[1,$\lfloor \frac{n}{2} \rfloor$ (G_1^*)],[$\lfloor \frac{n}{2} \rfloor + 1(G_1)$,n]

Construction of Sublines

To construct symmetric 2^i sublines, we mark cells of the position E_j with the following method.

$E_j = \{\sum_{i=0}^{j-1} 2^i, 2n - \sum_{i=0}^{j-1} 2^i\}$

(i) The position of the General is marked as E_1.

(ii) A collision cell of signals i and S_1 is marked as a righthand side of $\bar{E}_j$.

(iii) $E_j(2 <= j)$ is marked by collision of signals z and s_3 these two signals make collisions at all cells of $1, 3, 7, \cdots$.

(iv) Signals e_j are generated at each collision point of z and S_3, and cell at collision point of e and S_1 are righthand side of $E_j(2 <= j)$. and both E_j and $\bar{E}_j$ are symmetric.

Invoking 3n-time Solution

After constructing 2^i sublines, 3n-time solutions are applied to all regions of separated sublines. In our reversible solution, we use modified version of 99-state 3n-time based reversible solution in section 4.1.2. Several states and rules for invoking 3n-time solutions are added.

When signal S_1 reaches M_j, a 3n-time solution is invoked for the region of separated sublines. But central region do not always have the same length. So invoking timing has several subcases by the following property.

- the length of central region is 0 or not
- the number of cells in left-hand side of central region is odd or even
- the number of cells in right-hand side of central region is odd or even
- outside cells of the central region E_j or M_j

Construction by Reversible PCA

Partitioned cellular automata with layers

We use PCA to construct a solution as well as 3n-time one. Although this algorithm uses many signals and local rules seems to become very complex, the algorithm of making sublines and the part of 3n-time solution are independent except the position of invoking 3n-time solution (i.e., collision point of signals S_1 and M_j). So we use PCA with two layers LPCA(2) to construct a solution. Using LPCA(2) it is possible to construct rules for making sublines and for invoking 3n-time solution separately.

A one-dimensional three-neighbor PCA with two layers LPCA(2) LP is defined as follows,
$LP = (\mathbf{Z}, (P_0, P_1), (\tau_0, \tau_1), \varphi)$

- $\mathbf{Z}$ is the set of all integers at which cells are placed
- $P_0 = (\mathbf{Z}, (L_0, C_0, R_0), \varphi_0, q_0)$ is a PCA on layer 0
- $P_1 = (\mathbf{Z}, (L_1, C_1, R_1), \varphi_1, q_1)$ is a PCA on layer 1
- $\tau_0 : D_0 \to L_0 \times C_0 \times R_0 (D_0 \subset D)$ is a local transaction function between layers
- $\tau_1 : D_1 \to L_1 \times C_1 \times R_1 (D_0 \in D)$ is a local transaction function between layers, where $D = (R_0 \times C_0 \times L_0) \times (R_1 \times C_1 \times L_1)$
- $\varphi : D \to (L_0 \times C_0 \times R_0) \times (L_1 \times C_1 \times R_1)$ is a local function of LP defined as follows:
  $\forall x = ((r_0, c_0, l_0), (r_1, c_1, l_1)) \in D$
  (i) if $x \notin D_0$ and $x \notin D_1$ then $\varphi(x) = (\varphi_0(r_0, c_0, l_0), \varphi_1(r_1, c_1, l_1))$
  (ii) if $x \in D_0$ and $x \notin D_1$ then $\varphi(x) = (\tau_0(x), \varphi_1(r_1, c_1, l_1))$
  (iii) if $x \notin D_0$ and $x \in D_1$ then $\varphi(x) = (\varphi_0(r_0, c_0, l_0), \tau_1(x))$
  (iv) if $x \notin D_0$ and $x \notin D_1$ then $\varphi(x) = (\tau_0(x), \tau_1(x))$

But if the numbers of states of P_0, P_1 are large, one cannot expand the direct product in practice. So we made a simulation from $n = 1$ to 500 for checking reversibility for the following FSSP solution. i.e. We checked only used rules for this simulation were one-to-one.

$2n$-time Reversible Solution

We constructed a following LPCA(2) for FSSP. (This solution rapid protoytyping one and has $24 \times 42 \times 24 \times 4 \times 11 \times 4$ states.)

$LP = (\mathbf{Z}, (P_0, P_1), (\varphi_{01}, \varphi_{10}), \varphi)$

- $p_0 = (\mathbf{Z}, (L_0, C_0, R_0), \varphi_0,$ (“0”, “#”, “0”)),
  C_0 ={“#”,“0”,“a”,“b”,“c”,“d”,“e”,“G”,“G3”,“G2”,“Gk”,“E”,“M”,“F”,“A”,“B”,“C”,“D”, “Z”,“Y”,“I”,“A^ ”,“B^ ”,“C^ ”,“Z^ ”,“[”,“Mc”,“Md”,“Me”,“M~ ”,“Ec”,“Ed”,“Ee”, “Fc”,“Fd”,“Fe”,“Id”,“Ie”,“A-”,“C-”,“Z-”,“A~ ”},
  $L_0 = R_0$ ={“0”,“!”,“+”,“*”,“–”,“_”,“"”,“@”,“{”,“}”,“[”,“^ ”,“$”,“%”,“&”,“(”,“)”,“=”, “~ ”,“1”,“"”,“ ”,“.”,“/” }

- $p_1 = (\mathbf{Z}, (L_1, C_1, R_1), \varphi_1,$ (“#”, “#”, “#”)),
  C_1 ={“#”,“t”,“0r”,“0s”,“er”,“el”,“u”, “wr”,“wl”,“v”,“f” },

$L_1 = R_1 = \{$"0","+","*","-"$\}$

- General, soldier and quiescent state are ("0", "$G3$", " − ")$_1$("0", "wr", "0")$_2$, ("0", "0", "0")$_1$("0", "0r", "0")$_2$, and ("0", "#", "0")$_1$("0", "#", "0")$_2$ respectively.
- Firing state set: $F = (l_0, c_0, r_0) \times (l_1, f, r_1), (\forall l_0, c_0, r_0, l_1, r_1)$
- Transition rules $\varphi_0, \varphi_1, \varphi_{01}, \varphi_{10}$ are described in appendix A.

This solution is based on Mazoyer's $2n-2$ time FSSP construction without recursive calls. The layer 0 is used for constructing 2^i length sublines and the layer 1 for invoking Minsky solution. But due to using PCA, signal reflection takes 2 steps longer than usual CA. On this solution, constructing 2^i length sublines makes 2 step delays, and moreover, applying Minsky like solution makes 2 step delays, too. And thus it takes $2n+2$ firing time.

But it might be possible to construct $2n$ time solution on reversible PCA.

Fig.4.10 is the configuration of $n = 9$.

	a0R	3		s			s			s			s			s			s			s			s			s			s			s			s			s			s			s			s			s				
	a1R			11R	1		s			s			s			s			s			s			s			s			s			s			s			s			s			s			s			s				
	a2R	4		12R			s	1		s			s			s			s			s			s			s			s			s			s			s			s			s			s			s				
	a3R			21R	2		s			s	1		s			s			s			s			s			s			s			s			s			s			s			s			s			s				
	a4R			22R			11R			s			s	1		s			s			s			s			s			s			s			s			s			s			s			s			s				
	a5R			23R			12R			s			s			s	1		s			s			s			s			s			s			s			s			s			s			s			s				
	a6R			24R			s	2		s			s			s			s	1		s			s			s			s			s			s			s			s			s			s			s				
	a7R			25R			s			11R			s			s			s			s	1		s			s			s			s			s			s			s			s			s			s				
	W	a8		26R			s			12R			s			s			s			s			s	1		s			s			s			s			s			s			s			s			s				
	W			31R	4		s			s	2		s			s			s			s			s			s	1		s			s			s			s			s			s			s			s				
	W			32R			21R			s			11R			s			s			s			s			s			s	1		s			s			s			s			s			s			s				
	W			33R			22R			s			12R			s			s			s			s			s			s			s	1		s			s			s			s			s			s				
	W			34R			23R			s			s	2		s			s			s			s			s			s			s			s	1		s			s			s			s			s				
	W			35R			24R			s			s			11R			s			s			s			s			s			s			s			s	1		s			s			s			s				
	W			36R			25R			s			s			12R			s			s			s			s			s			s			s			s			s	1		s			s			s				
	W			37R			26R			s			s			s	2		s			s			s			s			s			s			s			s			s			s	1		s			s				
	W			38R			s	4		s			s			s			11R			s			s			s			s			s			s			s			s			s			s	1		s				
	W			39R			s			21R			s			s			12R			s			s			s			s			s			s			s			s			s			s			s	1		
	W			3AR			s			22R			s			s			s	2		s			s			s			s			s			s			s			s			s			s			s		b		
	W			3BR			s			23R			s			s			s			11R			s			s			s			s			s			s			s			s			s		1	b0L				
	W			3CR			s			24R			s			s			s			12R			s			s			s			s			s			s			s			s		1	s			b1L				
	W			3DR			s			25R			s			s			s			s	2		s			s			s			s			s			s			s		1	s			s		2	b2L				
	W			3ER			s			26R			s			s			s			s			11R			s			s			s			s			s		1	s			s			11L			b3L				
	W			s	8		s			s	4		s			s			s			s			12R			s			s			s			s		1	s			s			s			12L			21L	%			
	W			s			31R			s			21R			s			s			s			s	2		s			s			s		1	s			s			s			s		2	s			22L		%		
	W			s			32R			s			22R			s			s			s			s			11R			s		1	s			s			s			s			11L			s			23L	%			
	W			s			33R			s			23R			s			s			s			s			12R		1	s			s			s			s			s			12L			s			24L		%		
	W			s			34R			s			24R			s			s			s			s		c3	c0	c3		s			s			s			s			s		2	s			s			25L	%			
	W			s			35R			s			25R			s			s			s		c1	c11L			c0			c11R	c1		s			s			s			11L			s			s			26L		%		
	W			s			36R			s			26R			s			s		c1	s			c12L			c0			c12R			s	c1		s			s			12L			s			s		4	s	%			
	W			s			37R			s			s	4		s		c1	s			s		c2	s			c0			s	c2		s			s	c1		s		2	s			s			21L			s		%		
	W			s			38R			s			s		d1*	dR	d1+		s			c11L			s			c0			s			c11R			s		d1+	dL	d1*		s			s			22L			s	%			
	W			s			39R			s		d1	dL			dR			s	d1+		c12L			s			c0			s			c12R		d1+	s			dL			dR	d1		s			23L			s		%		
	W			s			3AR		d1	s			dL			dR			s			a21+			s			c0			s			a12+			s			dL			dR			s	d1		24L			s	%			
	W			s		3	a200	3		s			dL			dR			s		3	a210	3		s			c0			s		3	a120	3		s			dL			dR			s		3	a230	3		s		%		
	W		1	11L			a201			11R	1		dL			dR		1	11L			a211			11R	1		c0		1	11L			a121			11R	1		dL			dR		1	11L			a231			11R	1%			
	W	b		12L		4	a202	4		12R		b	dL			dR	b		12L		4	a212	4		12R		b	c0	b		12L		4	a122	4		12R		b	dL			dR	b		12L		4	a232	4		12R		b%		
	W		+	e12L	+		a203		+	e12Rb+			dL			dR		+	e12L	+		a213		+	e12Rb+			c0		+	e12L	+		a123		+	e12Rb+			dL			dR		+	e12L	+		a233		+	e12Rb%				
	fwR			f12L			f203			f12Rb			fdL			fdR			f12L			f213			f12Rb			fc0			f12L			f123			f12Rb			fdL			fdR			f12L			f233			f12Rb		%		

Figure 4.8: A configuration of a 2^3-divided solution (n=18)

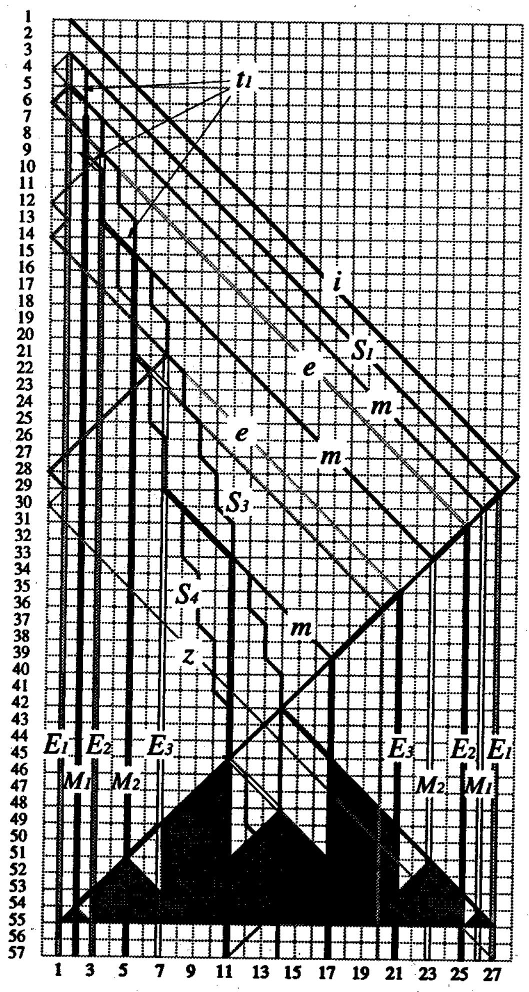


Figure 4.9: outline of $2n$-time solution by Goto and Mazoyer

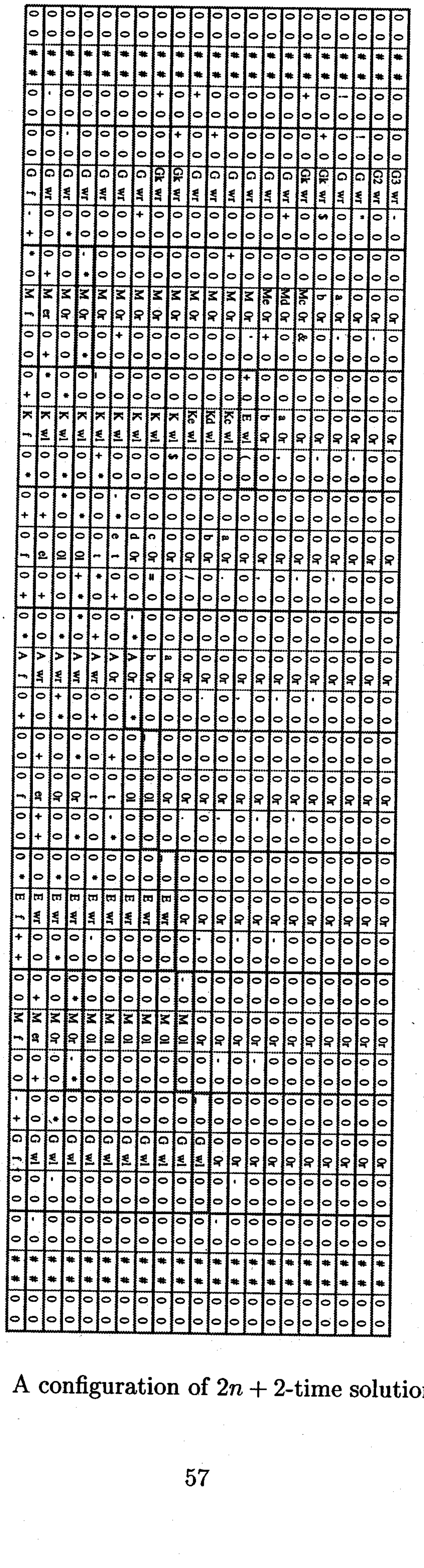


Figure 4.10: A configuration of $2n + 2$-time solution ($n = 9$)

4.1.5 Two-dimensional Cases

In this section, we construct reversible FSSP solutions to two-dimenstional connected figures.

Two cells $p = (x, y), p' = (x', y') \in \mathbf{Z}^2$ on a two-dimensional CA A are called *adjacent* if

$$(x = x' \wedge |y - y'| = 1) \vee (|x - x'| = 1 \wedge y = y')$$

If a sequence $p_0, p_1, \cdots, p_n, (p_0, p_1, \cdots, p_n \in \mathbf{Z}^2)$ satisfies $p_0 = p, p_n = p'$ and p_i, p_{i+1} are adjacent for all $i(0 \leq i < n)$, the sequence is called a *path* from p through p'. The integer n is the length of the path.

A subset $M \in \mathbf{Z}^2$ is *connected* if for any $p, p' \in M$, there is a path in M from p to p'. If M contains finite cells and also contains $(0, 0)$, we call M is a *figure.*

Synchronizing Condition for Two-Dimensional Reversible CA

In reversible CA, there is no solution with single firing state. So we define a synchronizing condition [RSC2] for two-dimensional reversible CA as well as one-dimensional case [35].

[RSC2] Let $A = (\mathbf{Z}^2, Q, \varphi_A, \#)$ be a two-dimensional reversible CA. A Synchronizing Condition for two-dimensional reversible CA is as follows.

Any figure $M(\subset Z^2)$, there exist two distinct states $g, s \in Q - \{\#\}$ and a state set $F \subset Q - \{\#, g, s\}$ that satisfy the followings.

1. $\forall u_1, u_2, u_3, u_4 \in \{s, \#\}$,
   $\varphi_A(s, u_1, u_2, u_3, u_4) = s$.

2. Let ${c_s}^{(M)}$ be a configuration defined by

$${c_s}^{(M)}(x) = \begin{cases} g & x = (0,0) \\ s & x \in M, x \neq (0,0) \\ \# & x \notin M \end{cases}$$

Then, there is a function t_f from subset of $\mathbf{Z}^2$ to natural number, that satisfies

$$\begin{aligned} &\forall x \in \mathbf{Z}^2 \\ &((x \in M \Rightarrow \Phi_A^{t_f(M)}({c_s}^{(M)})(x) \in F) \\ &\wedge (x \notin M \Rightarrow \Phi_A^{t_f(M)}({c_s}^{(M)})(x) \notin F)), \end{aligned}$$

$$\begin{aligned} &\forall i \in \mathbf{Z}_+(0 \leq i < t_f(M) \\ &\Rightarrow \forall x \in \mathbf{Z}^2, (\Phi_A^{t_f(M)}(c_s^{(M)})(x) \notin F)). \end{aligned}$$

This definition implies that we define a set $F(\subset Q)$ of finite number of firing states, and regard that the all cells synchronize if the state of each cell is in F at time $t = t_f(M)$. Moreover, we assume that not only the n cells but the other cells are allowed to change its internal states.

Realization of the Kobayashi's Solution on a Reversible PCA

Figure M_{step}

Kobayashi defined a class of figure $M_{step} \in M$ and constructed a minimum time solution for it [42]. In this section we first show a reversible FSSP solution for M_{step}. Although his main interest is to find faster synchronizing figures, we use it for its simplicity of assigning states. A Synchronizing sokution to any connected figures are disscussed in the next section.

1. Figure F contains one-left-down corner as (0,0) (the general cell)
2. F contains exactly one right-up corner
3. We can arrive at any cell starting form the left-down corner by properly proceeding up or to the right.
4. If we proceed up or to the right starting from a cell, we can eventually arrive at right up corner irrespective of the starting cell and the way we proceed
5. F contains no holes

We say the *length* of F is the Hamming distance between the general cell and right-up corner cell and denoted by *length*(F). If all soldier cells of F which has a same Hamming distance from the general have same behaviors, FSSP on F can be regarded as one-dimensional case of *length*(F).

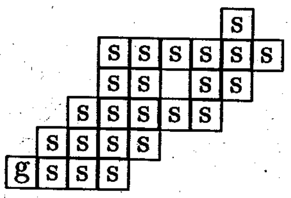


Figure 4.11: An example of M_{step}

A reversible solution for M_{step}

We constructed a reversible PCA in which Kobayashi's algoridhm and one-dimensional $3n$ time reversible solution was embedded.

All cells are categorized into 9 types(fig.4.12), As the result, the directions of up and right (left and bottom) of the each cells are regarded as the same behavior as in a solution of one dimensional cases.

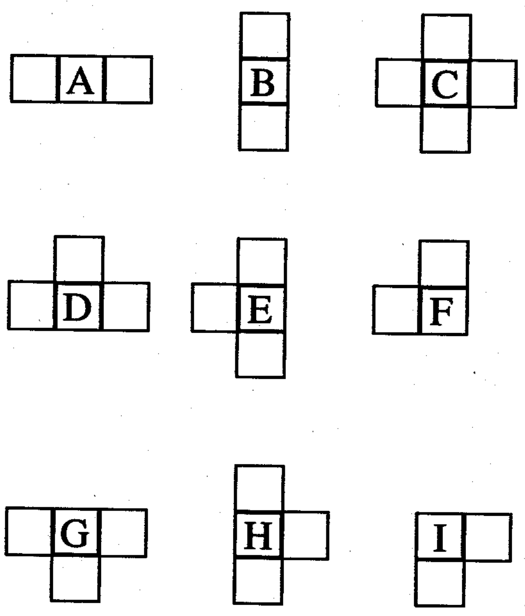


Figure 4.12: cell types

Following formula shows the solution we constructed. In the formula, like "(A, B)(a, b)" denotes "Aa, Ab, Ba, Bb" and parentheses are eliminated if they contain only one element.

- $P_K =$
  $(\mathbf{Z}^2, (C_K, U_K, R_K, D_K, L_K), \varphi_K, (\#, \#, \#, \#, \#)),$
  $C_K = \{\#, (A, B, C, D, E, F, G, H, I)(t, \overrightarrow{s}, \overleftarrow{s}, \overrightarrow{e}, \overleftarrow{e}, u, \overrightarrow{w}, \overleftarrow{w}, v, f, \acute{s}, \acute{e}, \acute{u}, \acute{w})\},$
  $U_K = R_K = D_K = L_K = \{\#, +, *, 1\}$

- General is $(\overrightarrow{As}, *, *, \#, \#)$, soldier is $(\overrightarrow{Cs}, \#, \#, \#, \#)$, quiescent state is $(\#, \#, \#, \#, \#)$.
- Firing state set contains all states which their center part are $(A, B, C, D, E, F, G, H, I)f$
- Local function φ_K is omitted (it can be found at the Web cite shown at the end of this section).

Next, we show the outline of this solution.

First, the general cell generates cell classification signal ("*") at $t = 0$. When the cell reveives a signal "*", it sends "*" signals to positive directions and if it is a soldier cell then it returns a signal "1" otherwise (i.e. a blank cell) it returns a signal "*" simultaneously (fig.4.14). Receiving signals from positive direction, the general cell determines its cell type and generates

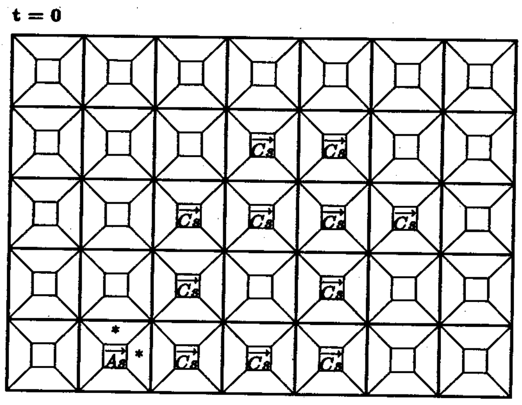


Figure 4.13: Configuration ($t = 1$)

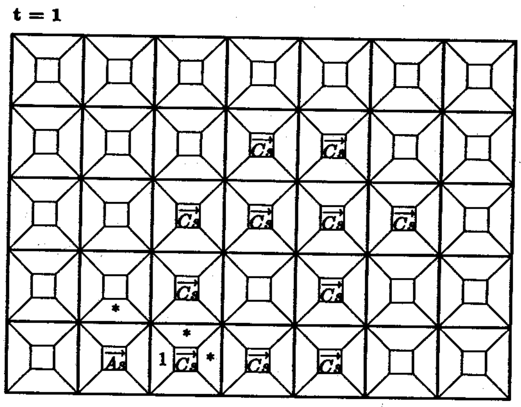


Figure 4.14: Configuration ($t = 2$)

speed 1 signal for synchronizing(fig. 4.15). All other soldier cells determine their cell types by returned signals and advancing speed 1 signals in turn (fig. 4.16).

Types of all cells are determined in the similar way and the signals of velocity 1 are transmitted. Most of all transition rules are direct product of transition rules P_1 and cell types, but keeping reversibility, a several rules should be changed.

The solution we constructed has 127 state for C part, 4 state for U, R, D, L part. The firing time $t_f = 3n - 1$ for a length n figure.

Realization of the Rosenstiehl's Solution on a Reversible CA

The algorithm of Rosenstiehl

FSSP solution for figures were shown by Rosenstiehl in 1966 [72], [42]. Firing time $t_f = 4N - 6$ for a figure of N cells. His algorithm is the following.

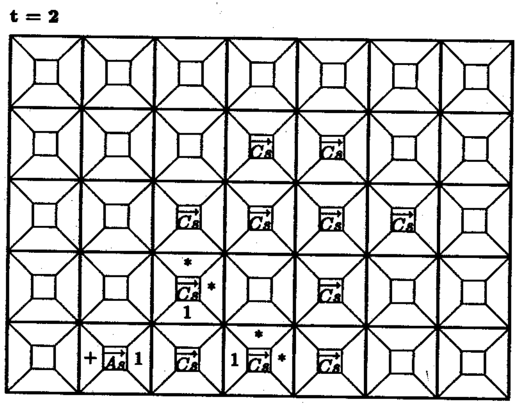


Figure 4.15: Configuration ($t = 3$)

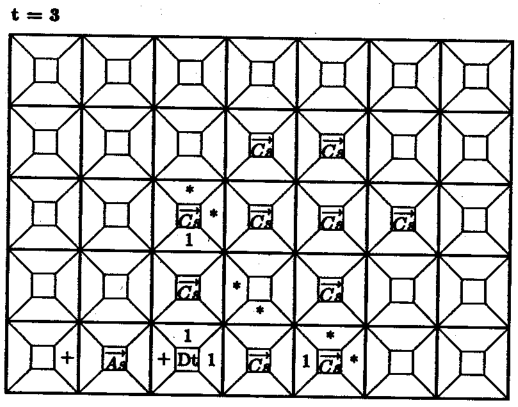


Figure 4.16: Configuration ($t = 4$)

Suppose $M \subset Z^2$ is any figure of n cells. We make a path which contains all cells of M by the following rules.

1. Let p be the cell (0,0) (i.e. the general cell). Go to (2).
2. If p has no adjacent cell which has not been visited then go to (3). Otherwise select one of such adjacent cells with the highest priority (Priority for p depends its direction right is the highest and up, left, and down follows). Set the selected cell as p and go to (2).
3. Let $p_1, \cdots, p_m$ are the adjacent cells of p then there exists exactly one p_i which has not been visited from p. If p is (0,0) and there is no adjacent cells which has not been visited, then stop, otherwise set p_i as p and go to (2).

By this algorithm, a path of $2N-2$ points is generated (Fig. 4.17). If one-dimensional $2N-2$ time solution is applied to this path, firing time of this figure M is $2(2N-2)-2 = 4N-6$.

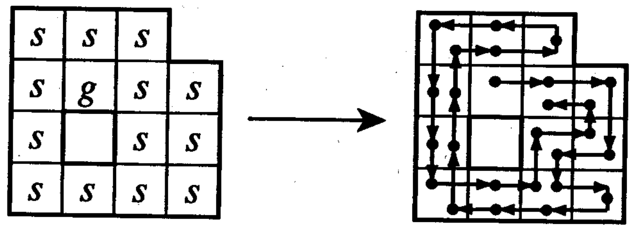


Figure 4.17: Rosenstiehl

A reversible Construction of Rosenstiehl's Solution

First we construct a reversible PCA P_R which constructs a path for any figure by the Rosenstiehl's algorithm.

- $P_R =$
  $(\mathbf{Z}^2, (C_R, U_R, R_R, D_R, L_R), \varphi_R, (\#, \#, \#, \#, \#))$,
  $C_R = \{\#, aa, ab3, ab34, ab4, aba, aba01, aba02, aba03, abca, abca01, abd3, abda, abda01, abda02, abdca, aca, ad3, ada, ada01, adca, ba3, ba34, ba4, bab, bab01, bab02, bab03, bacb, bacb01, bad3, badb, badb01, badb02, badcb, bb, bcb, bd3, bdb, bdb01, bdcb, ca4, cac, cac01, cadc, cb1, cb14, cb4, cba4, cbac, cbac01, cbac02, cbadc, cbc, cbc01, cbc02, cbc03, cbdc, cbdc01, cc, cdc, da3, dacd, dad, dad01, db1, db13, db3, dba3, dbacd, dbad, dbad01, dbad02, dbcd, dbcd01, dbd, dbd01, dbd02, dbd03, dcd, dd, g, g\#, ga, ga01, ga02, ga3, ga34, ga4, gac, gac01, gad, gad01, gad02, gad3, gadc, gb, gb01, gb02, gb03, gb04, gb05, gb06, gb1, gb13, gb134, gb14, gb3, gb34, gb4, gba, gba01, gba02, gba03, gba04, gba05, gba06, gba07, gba08, gba3, gba301, gba34, gba4, gba401, gbac, gbac01, gbac02, gbad, gbad01, gbad02, gbad03, gbad3, gbadc, gbc, gbc01, gbc02, gbc03, gbd, gbd01, gbd02, gbd03, gbd04, gbd05, gbd3, gbd301, gbdc, gbdc01, gd01, gd3, gdc, s\}$,
  $U_R = R_R = D_R = L_R = \{\#, ?, +, -, =\}$

- The transition rules of local function φ_R are omitted. (See the Web site described later.)

Next we explain the outline of this solution.

Realization of the Rosenstiehl's algorithm is classification of the all cells by the connected relation between each cells. This is the same approach as Kobayashi's algorithm, but it is much complex than Kobayashi's algorithm.

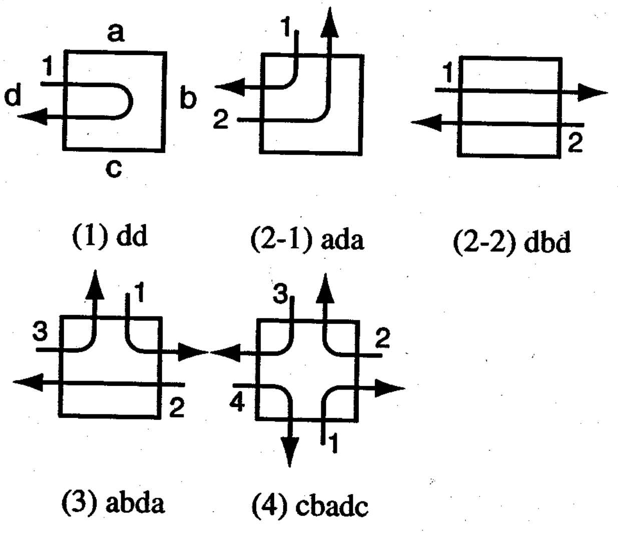


Figure 4.18: the codings of connection of a path(soldier cell)

Four directions "up", "right", "down", "left" are encoded by symbols "a","b","c","d" (alphabetical expression) or "1","2","3","4" (numerical expression) respectively. A type of a cell is denoted by a string consists of direction alphabets which show the directions of entering/outgoing points of path.

Because the number of the entering points of path for a cell is at most 4, type encoded strings for a soldier cell are classified into 5 categories 32 types. Length of all cell types l must satisfies $2 \leq l \leq 5$ and first and last alphabet should be the same. The general cell denoted as a catenation of "g" and direction alphabets.

At $t = 0$, the general generates inspection signals "?" (Fig. 4.19(b)). At $t = 1$, If the neighbor cell which receives signal "?" is a soldier cell then it returns signal "+", otherwise (i.e. a blank cell) it returns signals "−" (Fig. 4.19(c)).

At $t = 2$, the general changes its center state by the returned signals. On this example, "up" and "right" cell returns "+" signals. "Right" direction has higher priority than "up" direction, so the path outgoing to "right" direction. the general generates "=" signal to go to the next cell and changes its center part to "gb1" (Fig. 4.19(d)). "b" means that the path is connected to the "right" cell and "1" means that "up" direction has not been visited.

All other soldier cells update their cell type as the same method and until path is completed. We shows an example for explanation.

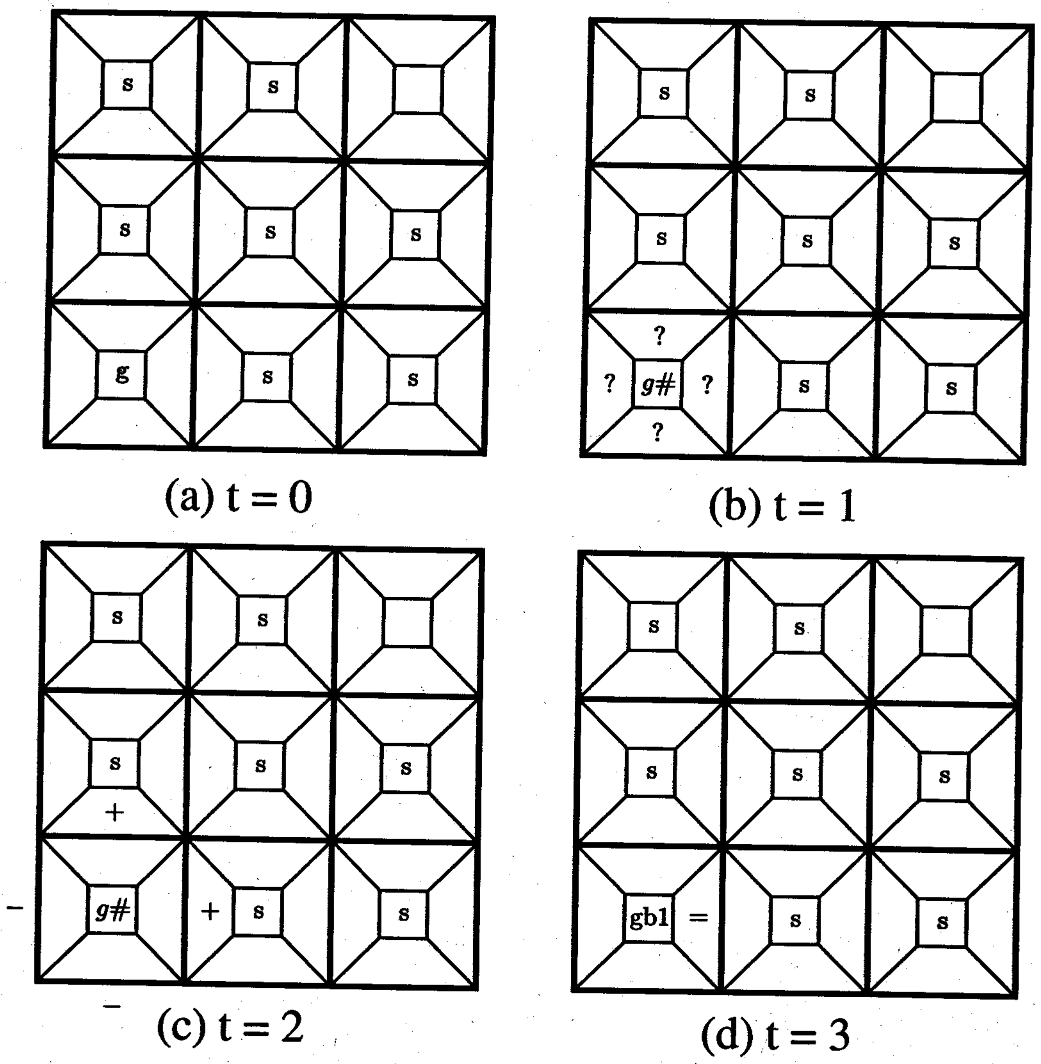


Figure 4.19: Start generating a path.

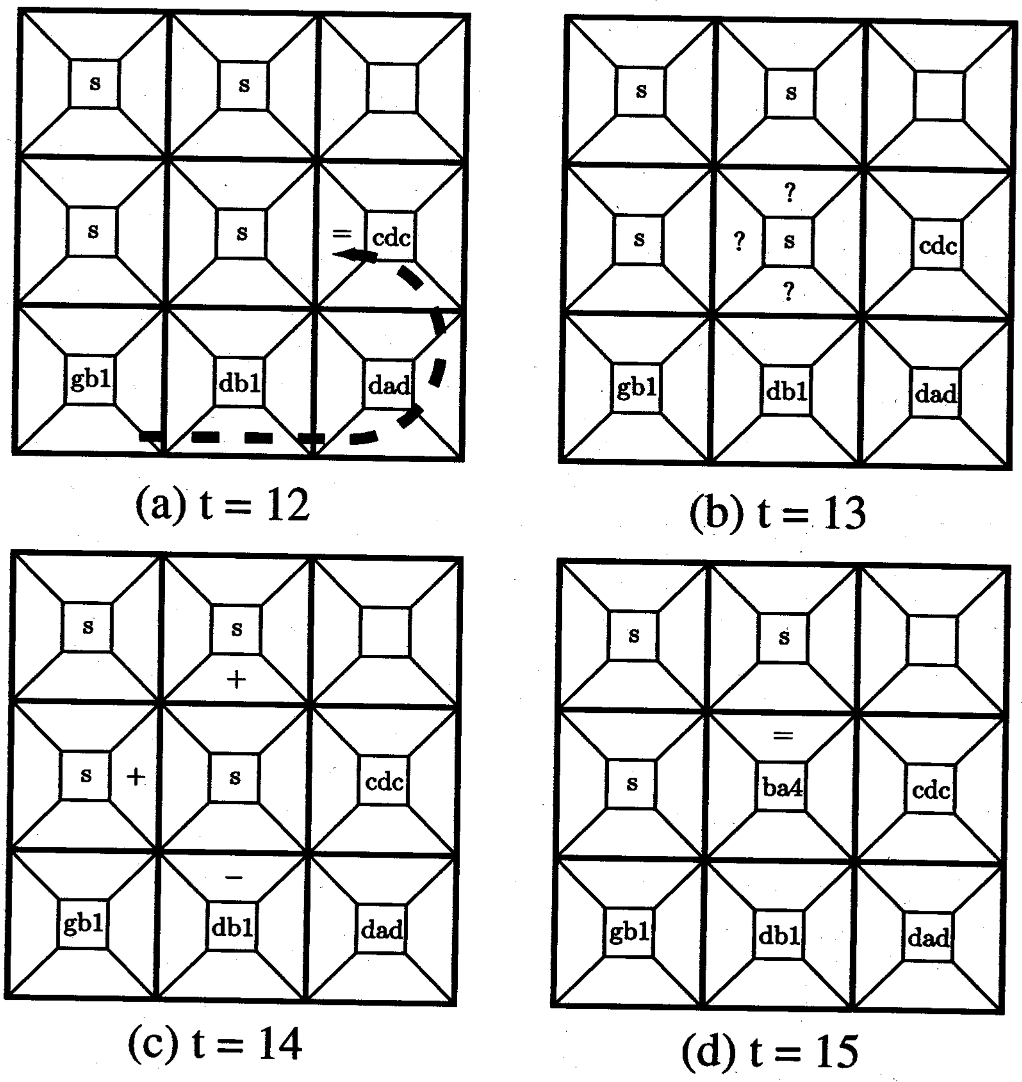


Figure 4.20: Determiing process of cell types

We make a brief explanation about a center soldier in this figure. When the cell received "=" signal from "b" direction (Fig. 4.20(a)), it generates inspection signals "?" (b). If the neighbor cell which receives signal "?" is a soldier cell then it returns signal "+", otherwise (i.e. a blank cell or already tested soldier cell) it returns signals "−" (Fig. 4.20(c)). The cell changes its center state by the returned signals and generates "=" signal to go to the next cell (d). We need 3 steps to determine the next cell. This is because we are using PCA.

In this manner, after all cell types are confirmed and the signal "=" has returned the general, path generation is finished. The states of center part of all cells on the figure are changed to cell type strings (Fig. 4.21). In Fig. 4.21, some state strings include the number starting with

"0", these state should be needed for keeping injectivity of local function. After all, we need $3(2N - 1)$ steps to complete a path for a figure of N cells.

The reversiblility of P_R is concluded by fact that φ_{P_R} is one-to-one (tested by computer).

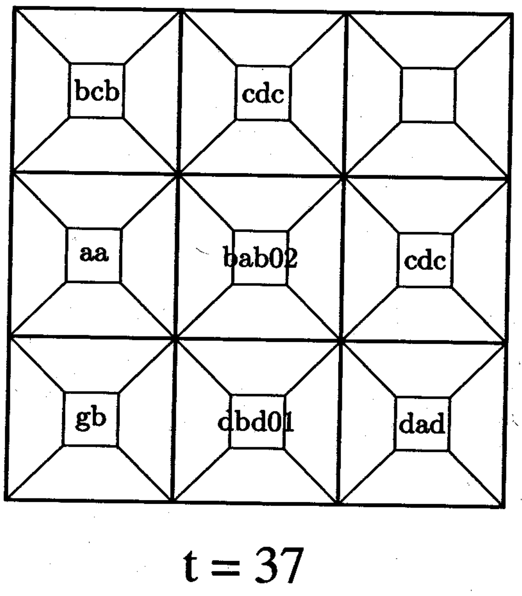


Figure 4.21: Configuration for confirmed Rosenstiehl's path

Invoking One-Dimensional Solution along a Rosenstiehl's Path

Direct products of all rules of reversible $3n$-time solution and all cell types for Rosenstiehl path leads to the solution. But many cell states are needed and it is very difficult to assign true states.

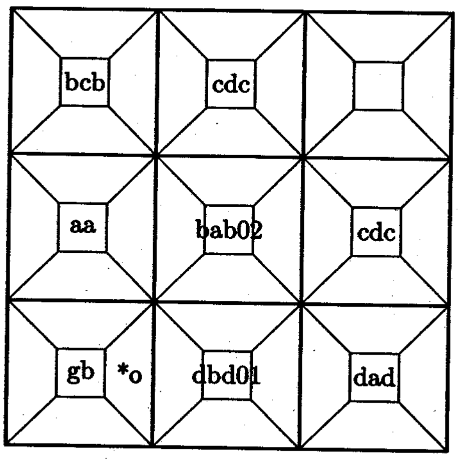
So, in this section, we constructed a solution P_{R2} which can synchronize connected figures with no branching Rosenstiehl path. Figures should be combined with the cells which has the number of input less than 2 such as (a),(b) in fig. 4.18.

Configurations of P_{R2} for a example figure is depicted in fig. 4.22.

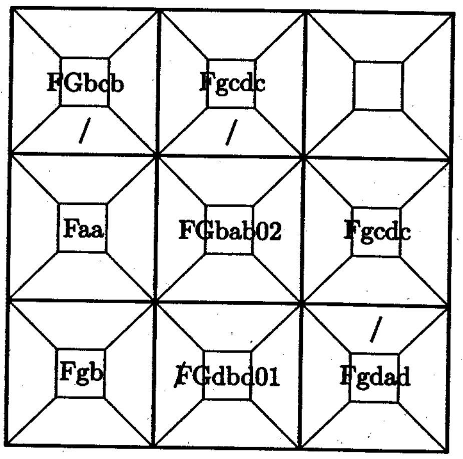
(a) is the configuration of the time that velocity 1 ("*") and velocity 1/3 ("o") signals are emitted after completed path (fig. 4.21). (b) is the configuration of firing time.

Firing time t_f is $3(2N - 1) + 3(2n - 2)$ when the method for emitting $3n$-time firing signal after completing Rosenstiehl's path is used. But no branching case such as P_{R2} is equal to the firing time for N cells.

Transitoin rules and detail explanation of P_{R2} is omoitted. Transition rules and some configurations are available at



(a) t = 37



(b) t = 55 firing

Figure 4.22: Configuration to a no-branching figure

`http://kelp.ke.sys.hiroshima-u.ac.jp/projects/rca/fssp/`

4.1.6 Summary

In this section, we constructed following reversible solutions to the FSSP. The numbers of their states are not optimal and more improvements are possible.

Table 4.5: Constructed reversibel solutions to the FSSP.

	used algorithm	sync. time	the number of states
1D	$3n$-time	$3n$	11×3^2
	2^2-divided	$7/3n$	40×9^2
	2^3-divided	$7/3n$	$44 \times 288 \times 40$
	Goto and Mazoyer based	$2n+2$	$24 \times 42 \times 24 \times 4 \times 11 \times 4$
2D	M_{step}		127×4^4
	Rosenstiel's algorithm		51×5^4

4.2 Number Conserving Solutions for FSSP

4.2.1 Number-Conserving Partitioned Cellular Automata

As we explained in section 2.4 the bit-conserving property is one of the most important property for the BBM.It is based on the analogy of mass conserving property on a physical system.

Although the bit-conserving property is defined on a 2-state CA, one can extend it into multi-bit cases. In this section, we define the number-conserveness for CA as follows,

Definition 4.2.1 A one dimensional deterministic CA $A = (\mathbf{Z}, Q, N, \varphi_A, \tilde{q})$, with global function Φ_A is said to be *number-conserving*, if A has following properties.

- $Q = \mathbf{N}_m, \mathbf{N}_m = \{0, 1, \cdots, m\}$
- $\forall c, \sum \Phi_A(c)(i) = C$, C is an integer.

There are several models such as sand pile [27] which based on the notion of number-conserving. But they are very special CA and they have very limited rules.And synchronizing problems are not studied on these models. Because, in general, it is very difficult to design a CA with number-conserving rules as well as reversible rules and as well as reversibility, number-conserveness is defined as the property related to its global function. So it is very difficult to determine whether CA is number-conserving or not.

But conserveness of local maps of PCA is useful for constructing number-conserving rules. i.e. its cell is divided into three parts, and hence each cell is represented by a triple of non-negative integers. Only neighboring parts are concerned in evolving each cell's state. So we use the framework of PCA to construct number-conserving CA. Number-conserving PCA [65] is defined in the same way.

Definition 4.2.2 A PCA P_{nc},

$$P_{nc} = (\mathbf{Z}, (L, C, R), \varphi_P, (\tilde{q}_L, \tilde{q}_C, \tilde{q}_R))$$

is said to be number-conserving, if following conditions holds,

- $L = C = R = \mathbf{N}_m, \mathbf{N}_m = \{0, 1, \cdots, m\}$.
- Quiescent state is $(\tilde{q}_L, \tilde{q}_C, \tilde{q}_R) = (0, k, 0), (0 < k \leq m)$.
- For all (r, c, l) and $(l', c', r') \in \mathbf{N}_m^3$, if $\varphi_{P_{nc}}(r, c, l) = (l', c', r')$ then $r + c + l = l' + c' + r'$.

In NC-PCA, only the constraint that the local transition function should satisfy the number-conserving condition is supposed. Thus, it makes relatively easy to construct an NC-PCA. Because each cell can hold three non-negative integers, it is possible to represent different states even if the sum of three numbers are equal.

In this section, we show the result of constructing the number-conserving solution to the Firing Squad Synchronization Problem [39].

4.2.2 FSSP Conditions for Number-Conserving Solutions

In a number-conserving cellular space, propagation of signals are formed by moving some number to neighboring cells, and global transfer of signals change the balance of numbers. On FSSP, all cells must be synchronized into a single state (i.e., the same number) using only local rules, so all signals used for algorithms to FSSP on number-conserving CA use signals should be "balanced". $3n$-time algorithm is based on a divide-and-conquer method. It is possible to realize FSSP solutions on NC-PCA, if its transition rules are constructed to keep the sum of each side of cell states are equal number at all points of generating new generals.

As mentioned in section 4.1.1, on reversible CA, it is impossible to construct the solution to FSSP even if framework of PCA is used. But on NC-PCA, it can be possible to construct the solution based on a single firing state $f = (f_l, f_c, f_r)$. But because we use the framework of PCA, in order to recognize the right most soldier cell, a cell state of right wall should be changed, so the second constraint of RSC in section 4.1.1 is still remained.

Synchronizing Conditions for Number-Conserving Solution (NCSC) All condition is the same as RSC in section 4.1.1, except single firing state $f = (f_l, f_c, f_r)$.

4.2.3 A Solution on Number-Conserving PCA

Under the synchronizing condition denoted above, we constructed a number-conserving solution P_{ncfssp}.

$$P_{ncfssp} = (\mathbf{Z}, (9,9,9), \varphi_{ncfssp}, (0,0,0))$$

general state is $g = (1,7,1)$, soldier state is $s = (0,9,0)$, firing state is $f = (2,5,2)$ and local function φ_{ncfssp} is composed by the listed local transition rules in Fig.4.24.

Configuration of P_{ncfssp} $(n = 8)$ is depicted in Fig.4.23 We briefly denoted the outline of this solution.

This solution is based on $3n$-time algorithm. Rules $\{1\}$ are used for the static states. Signals of velocity 1 and -1 are denoted by the rule $\{2\}$, $\{3\}$ and signals of velocity $\pm 1/3$ uses rules $\{5\}$. $\{6\}$, $\{7\}$ are rules for collisions and final transitions for firing state are denoted by rules $\{8\}$.

Differ from reversible PCA, Although NC-PCA allows the existence of same states in the right hand side of its transition rules, the following type of transitions: $(0,k,0) \to (0,l,0), (k \neq l)$ are inhibited. This type of rules are usually used to implement counters, but realization of these counters (i.e., a signal delay feature) with a single cell is impossible. When we implement a signal of velocity $1/3$, it is impossible to make a delay only a center part of a PCA cell. We must send a signal to a neighboring cell and receive its reflection signal to realize a delay.

On reversible solution in section 4.1.2, we uses the state for preparing to fire denoted by the symbol e. Preparing symbol e can be autonomously changed to firing state. In contrast to reversible solutions, preparing configurations for a number-conserving solution are quite confusing. Because in its preparing stage, the numbers are completely distributed to each partitions for its firing configurations.

When cells are divided into two parts by generating new generals, the sum of states should be equal number. A signal of velocity 1 is described by the state 1 and it is generated by decrement the state number of a general. So after the collision of signals on the center cell, the signal of velocity -1 is encoded by state 2 and the number of general cell can be recovered.

This solution is $m = 9$ but it might be possible to find smaller solutions. Although this solution is based on $3n$-time algorithm, a $2n + c$-time solution will be available if Goto and Mazoyer algorithm [50] is used.

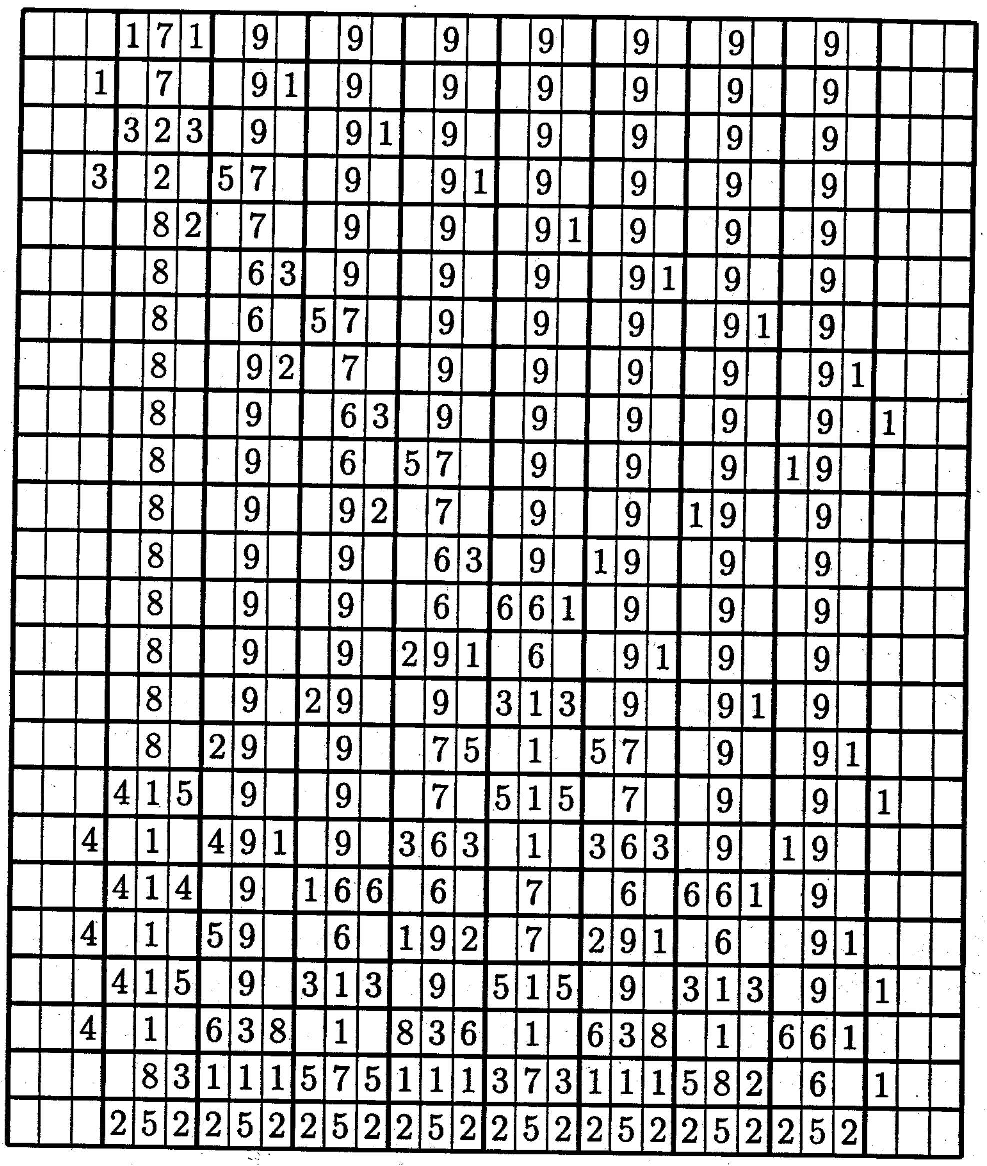


Figure 4.23: A configuration of P_{ncfssp} (n=8)

{1} $(0,0,0) \to (0,0,0)$, $(0,1,0) \to (0,1,0)$, $(0,2,0) \to (0,2,0)$,
$(0,6,0) \to (0,6,0)$, $(0,7,0) \to (0,7,0)$, $(0,8,0) \to (0,8,0)$,
$(0,9,0) \to (0,9,0)$,

{2} $(1,9,0) \to (0,9,1)$, $(0,9,1) \to (1,9,0)$, $(2,9,0) \to (0,9,2)$,
$(0,9,2) \to (2,9,0)$, $(4,9,1) \to (5,9,0)$, $(1,9,4) \to (0,9,5)$,

{3} $(1,0,0) \to (1,0,0)$, $(0,0,1) \to (0,0,1)$, $(0,8,2) \to (4,1,5)$,
$(2,8,0) \to (5,1,4)$, $(5,9,0) \to (4,9,1)$, $(0,9,5) \to (1,9,4)$,
$(4,1,5) \to (4,1,5)$, $(5,1,4) \to (5,1,4)$, $(2,7,2) \to (5,1,5)$,

{4} $(0,0,4) \to (0,0,4)$, $(4,0,0) \to (4,0,0)$, $(4,1,4) \to (4,1,4)$,
$(4,9,0) \to (4,9,0)$, $(0,9,4) \to (0,9,4)$,

{5} $(3,9,0) \to (5,7,0)$, $(0,9,3) \to (0,7,5)$, $(0,6,5) \to (0,9,2)$,
$(5,6,0) \to (2,9,0)$, $(2,7,0) \to (0,6,3)$, $(0,7,2) \to (3,6,0)$,
$(1,7,0) \to (3,2,3)$, $(0,7,1) \to (3,2,3)$, $(0,0,3) \to (0,0,3)$,
$(3,0,0) \to (3,0,0)$, $(3,2,5) \to (0,8,2)$, $(5,2,3) \to (2,8,0)$,

{6} $(3,9,1) \to (6,6,1)$, $(1,9,3) \to (1,6,6)$, $(0,6,6) \to (2,9,1)$,
$(6,6,0) \to (1,9,2)$, $(1,6,0) \to (3,1,3)$, $(0,6,1) \to (3,1,3)$,
$(5,1,5) \to (5,1,5)$, $(0,7,5) \to (3,6,3)$, $(5,7,0) \to (3,6,3)$,
$(3,1,3) \to (0,7,0)$,

{7} $(2,7,1) \to (0,4,6)$, $(1,7,2) \to (6,4,0)$, $(5,7,1) \to (3,4,6)$,
$(1,7,5) \to (6,4,3)$, $(0,4,0) \to (2,0,2)$, $(6,9,0) \to (8,6,1)$,
$(0,9,6) \to (1,6,8)$, $(6,9,4) \to (8,6,5)$, $(4,9,6) \to (5,6,8)$,
$(0,0,8) \to (0,2,6)$, $(8,0,0) \to (6,2,0)$, $(2,6,0) \to (6,2,0)$,
$(0,6,2) \to (0,2,6)$, $(0,2,6) \to (3,4,1)$, $(6,2,0) \to (1,4,3)$,
$(0,4,1) \to (0,4,1)$, $(1,4,0) \to (1,4,0)$, $(5,4,1) \to (2,1,7)$,
$(1,4,5) \to (7,1,2)$, $(0,1,7) \to (0,8,0)$, $(7,1,0) \to (0,8,0)$,

{8} $(5,9,3) \to (6,3,8)$, $(3,9,5) \to (8,3,6)$, $(0,3,0) \to (1,1,1)$,
$(4,1,6) \to (0,8,3)$, $(6,1,4) \to (3,8,0)$, $(8,1,8) \to (5,7,5)$,
$(6,1,6) \to (3,7,3)$, $(8,1,6) \to (5,8,2)$, $(6,1,8) \to (2,8,5)$,
$(0,8,1) \to (2,5,2)$, $(1,8,0) \to (2,5,2)$, $(3,1,5) \to (2,5,2)$,
$(5,1,3) \to (2,5,2)$, $(1,7,1) \to (2,5,2)$, $(2,6,1) \to (2,5,2)$,
$(1,6,2) \to (2,5,2)$, $(0,1,4) \to (0,1,4)$, $(4,1,0) \to (4,1,0)$,
$(5,2,6) \to (6,3,4)$, $(6,2,5) \to (4,3,6)$, $(6,2,1) \to (2,7,0)$,
$(1,2,6) \to (0,7,2)$, $(0,3,4) \to (1,1,5)$, $(4,3,0) \to (5,1,1)$,
$(0,3,2) \to (1,1,3)$, $(2,3,0) \to (3,1,1)$, $(4,7,0) \to (5,6,0)$,
$(0,7,4) \to (0,6,5)$, $(3,6,0) \to (2,5,2)$, $(0,6,3) \to (2,5,2)$,
$(8,4,1) \to (5,7,1)$, $(1,4,8) \to (1,7,5)$, $(1,4,6) \to (1,8,2)$,
$(6,4,1) \to (2,8,1)$, $(3,2,6) \to (0,3,8)$, $(6,2,3) \to (8,3,0)$,
$(8,6,1) \to (6,2,7)$, $(1,6,8) \to (7,2,6)$, $(0,1,6) \to (0,7,0)$,
$(6,1,0) \to (0,7,0)$, $(1,2,0) \to (1,2,0)$, $(0,2,1) \to (0,2,1)$,
$(7,0,0) \to (7,0,0)$, $(0,0,7) \to (0,0,7)$, $(0,2,7) \to (2,5,2)$,
$(7,2,0) \to (2,5,2)$

Figure 4.24: The local rule of P_{ncfssp}

Chapter 5

Self-Reproduction in Reversible Cellular Automata

5.1 Self-Reproduction in a Two-Dimensional RPCA

5.1.1 Definition of SR_8

In this chapter, we construct non-trivial self-reproducing structures can be constractible in a reversible cellular space.

The idea of our model is based on Langton's sheathed Loop, and to achieve more flexibility we introduced a self-inspection method. Although Ibáñes et al. [32] showed a 16-state model in which sheathed loops can reproduce by using a self-inspection method independently, our model is realized in a "reversible" cellular space.

In the cellular space of SR_8, 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. By using these operations, various objects called Worms and Loops can reproduce themselves in a very simple manner.

The RPCA "SR_8" is defined by

$$SR_8 = (\mathrm{Z}, (C, U, R, D, L), g, (\#, \#, \#, \#, \#)),$$
$$C = U = R = D = L = \{\#, *, +, -, \mathrm{A,B,C,D}\}.$$

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 contains 765 rules, and is a one-to-one mapping. The complete listing of the rules is given in Appendix.B.

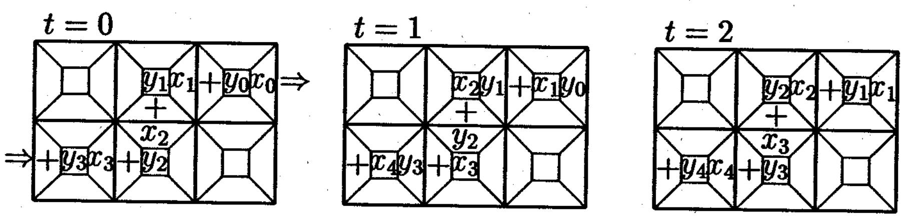


Figure 5.1: Signal transmission on a part of a simple wire $(x_i, y_i \in \{\text{A,B,C}\})$.

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)

Table 5.1: Six commands composed of A, B, and C.

5.1.2 Signal Transmission on a Wire

A *wire* is a configuration to transmit signals A, B, and C. Fig. 5.1 shows an example of a part of a simple (i.e., non-branching) wire.

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 5.1. These commands are used for extending or branching a wire.

5.1.3 A Worm

A *Worm* is a simple wire with open ends that are called a *head* and a *tail*. It crawls in the reversible cellular space as shown in Fig. 5.2.

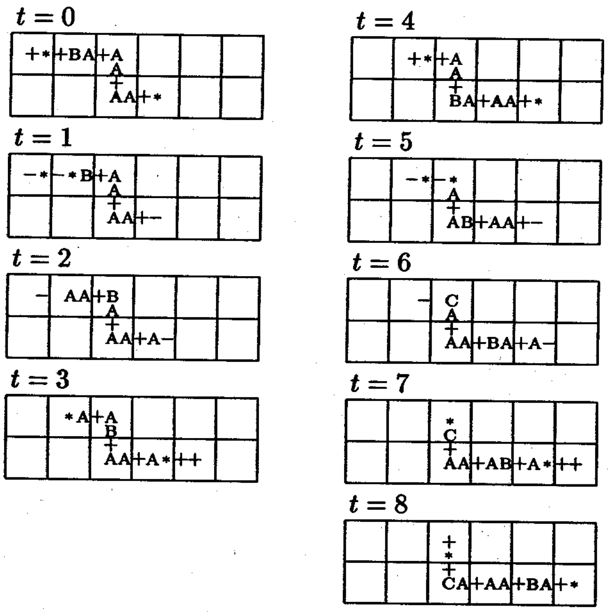


Figure 5.2: Behavior of a Worm.

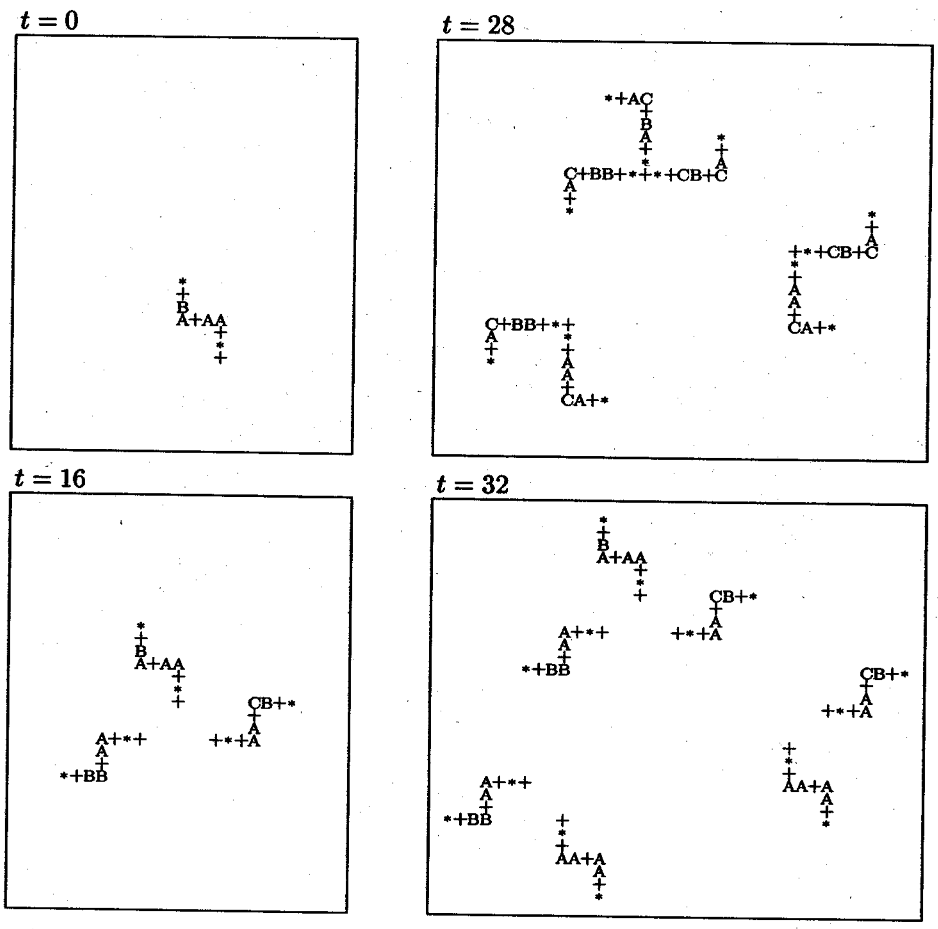


Figure 5.3: Self-reproducing process of a Worm.

Commands in Table 5.1 are decoded and executed at the head of a Worm. That is, the command AA extends the head straight, while the command AB (or AC, respectively) extends it leftward (rightward). 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.

5.1.4 Self-Reproduction of a Worm

By giving a branch command, *any* Worm can self-reproduce indefinitely provided that it neither cycles nor touches itself in the branching process. Fig. 5.3 shows self-reproducing processes of Worms.

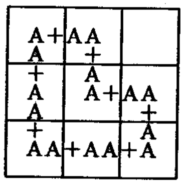


Figure 5.4: An example of a Loop.

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

Table 5.2: Commands DB and DC.

5.1.5 Self-Reproduction of a Loop

A *Loop* is a simple closed wire, thus has neither a head nor a tail as shown in Fig. 5.4.

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 5.2 are used.

Examples of entire self-reproducing processes of Loops are shown in Figs. 5.5 and 5.6. By putting a command DB at an appropriate position, *every* Loop having only AA commands in all the other cells can self-reproduce in this way. When DB reaches the bottom right corner, it starts making an "arm" and this corner become a transmitter of commands. First, all AA commands in the mother Loop are transmitted through the arm and generated DC commands encode whole shape of the mother Loop into command sequences simultaneously and these commands are transmitted after all static AA commands are transmitted. Finally DC commands reaches the bottom right corner and the arm is splitted from the mother Loop.

5.1.6 Controlling the Position of Daughter Loops in SR_8

One of our main motivations is to place preferred initial patterns to a reversible cellular space. As mensioned above, a closed Loop has only AA commands. If AB or AC commands are placed in the Loop, generated positon of the daughter Loop can be cahnged.

But DB (create an arm command) advaces the bottom side of a loop and the length of the Loop does not equal to the running length of the whole commands. Thus the embedded

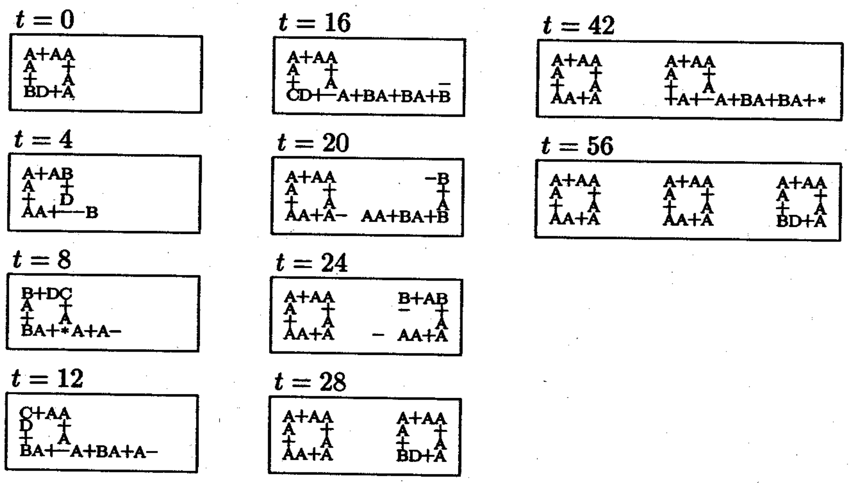


Figure 5.5: Self-reproducing process of a Loop (1)

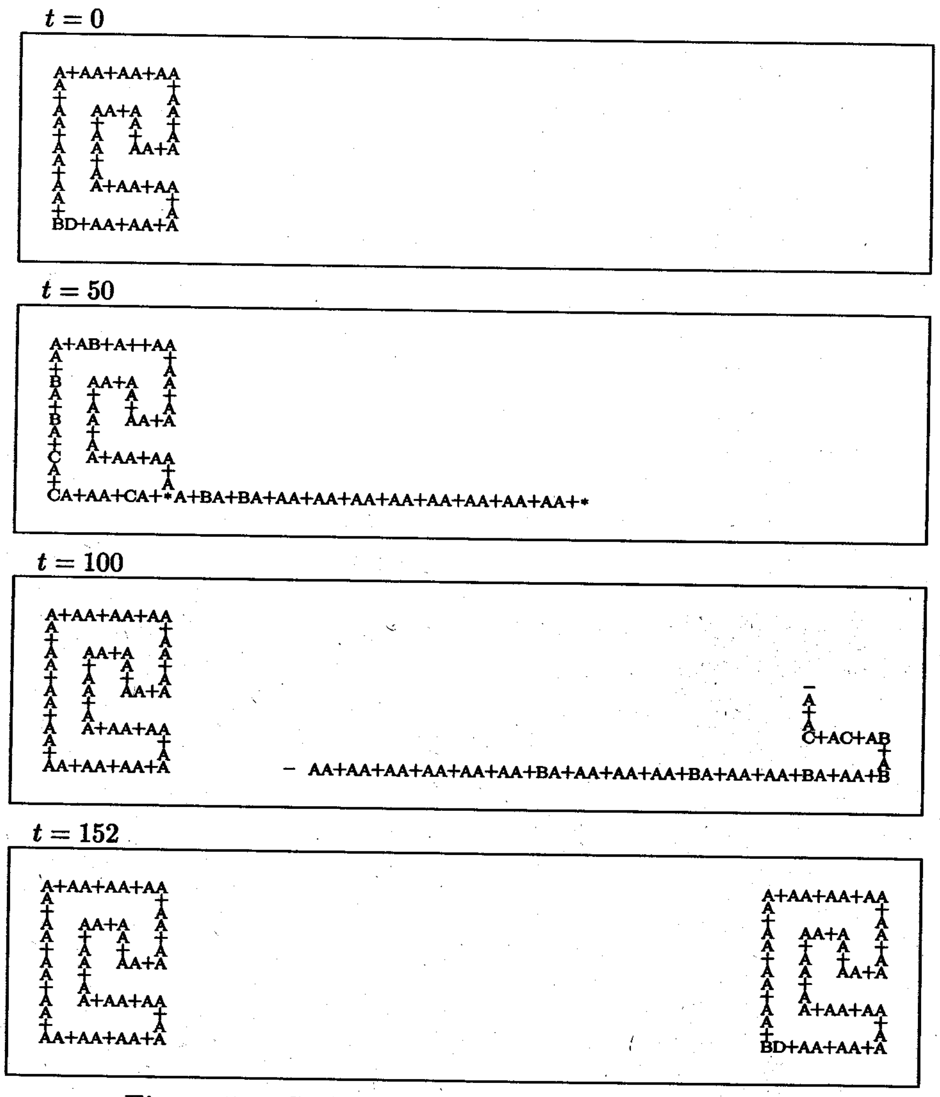


Figure 5.6: Self-reproducing process of a Loop (2)

position of turning commands in the daughter Loop differ from the mother Loop (Fig.5.7).

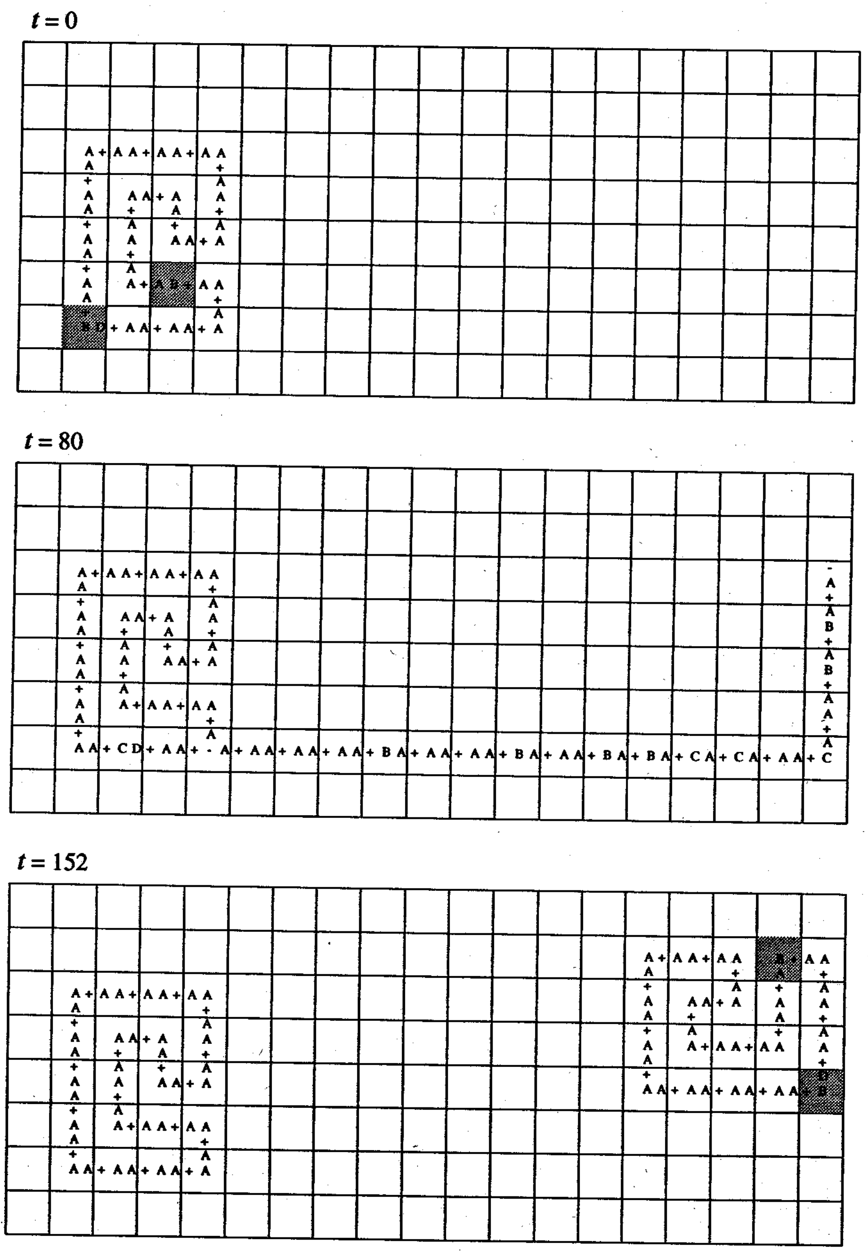


Figure 5.7: A shifting command sequence by the self-reproducing process of a Loop of SR_8

Although such a shifting of reading-frames of its command sequence is interesting phenomenon, it is difficult to control. So we modify SR_8 for solving this timing problem.

Fig.5.8 is the process of modified version of SR_8. DB signal is not advance bottom side and reproducing starts from the bottom right corner as soon as the Worm reaches at this position. So created daughter Loop is rotated in 90 degrees. Because of this rotation, Loops make collision after 4 generations. But this collision can be avoidable by inserting direction commands into the mother Loop (Fig.5.9) and this modification acts important roll in extending SR_8 to three-dimensional one in the next section.

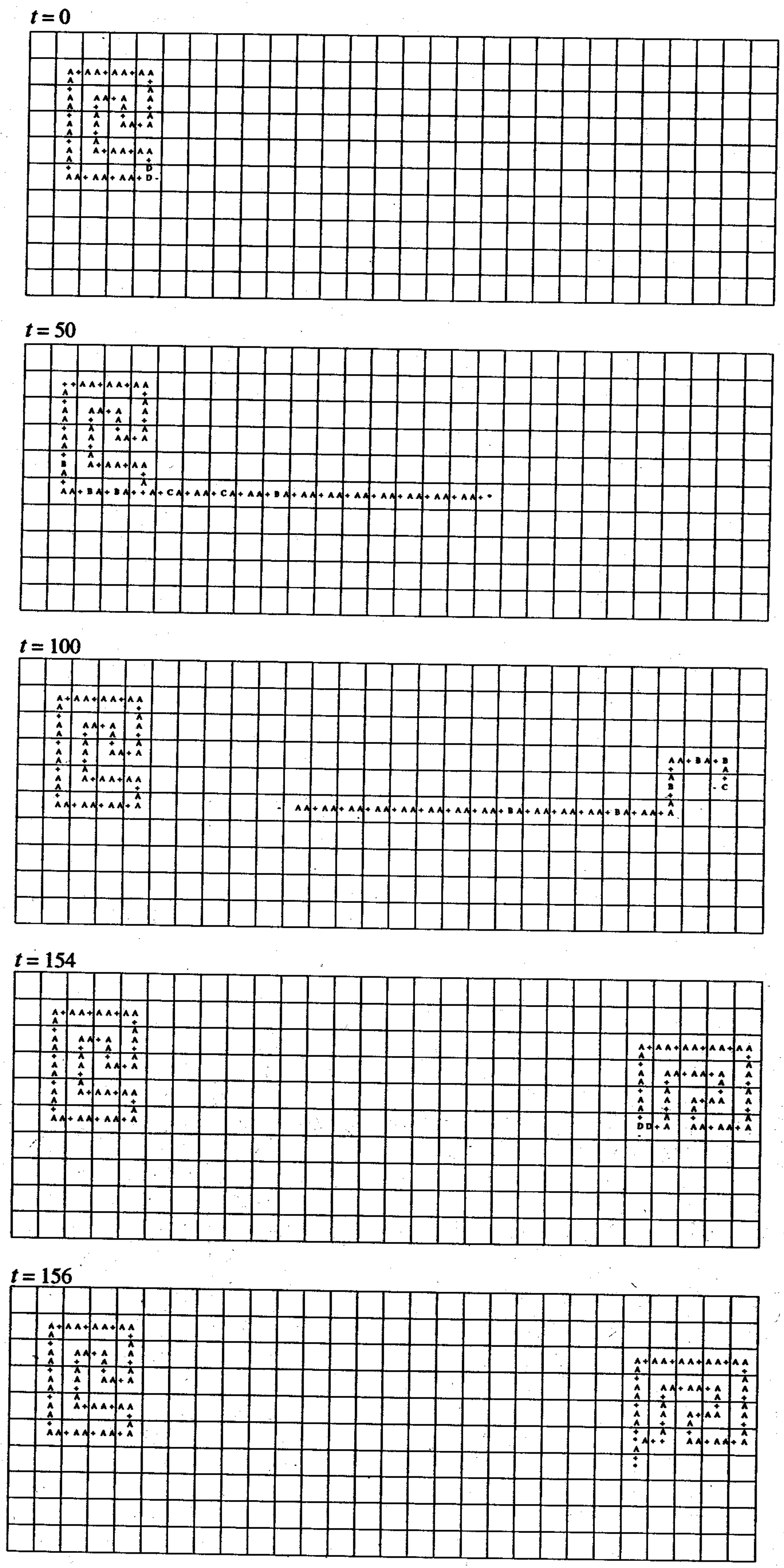


Figure 5.8: Self-reproducing process of a Loop of modified SR_8

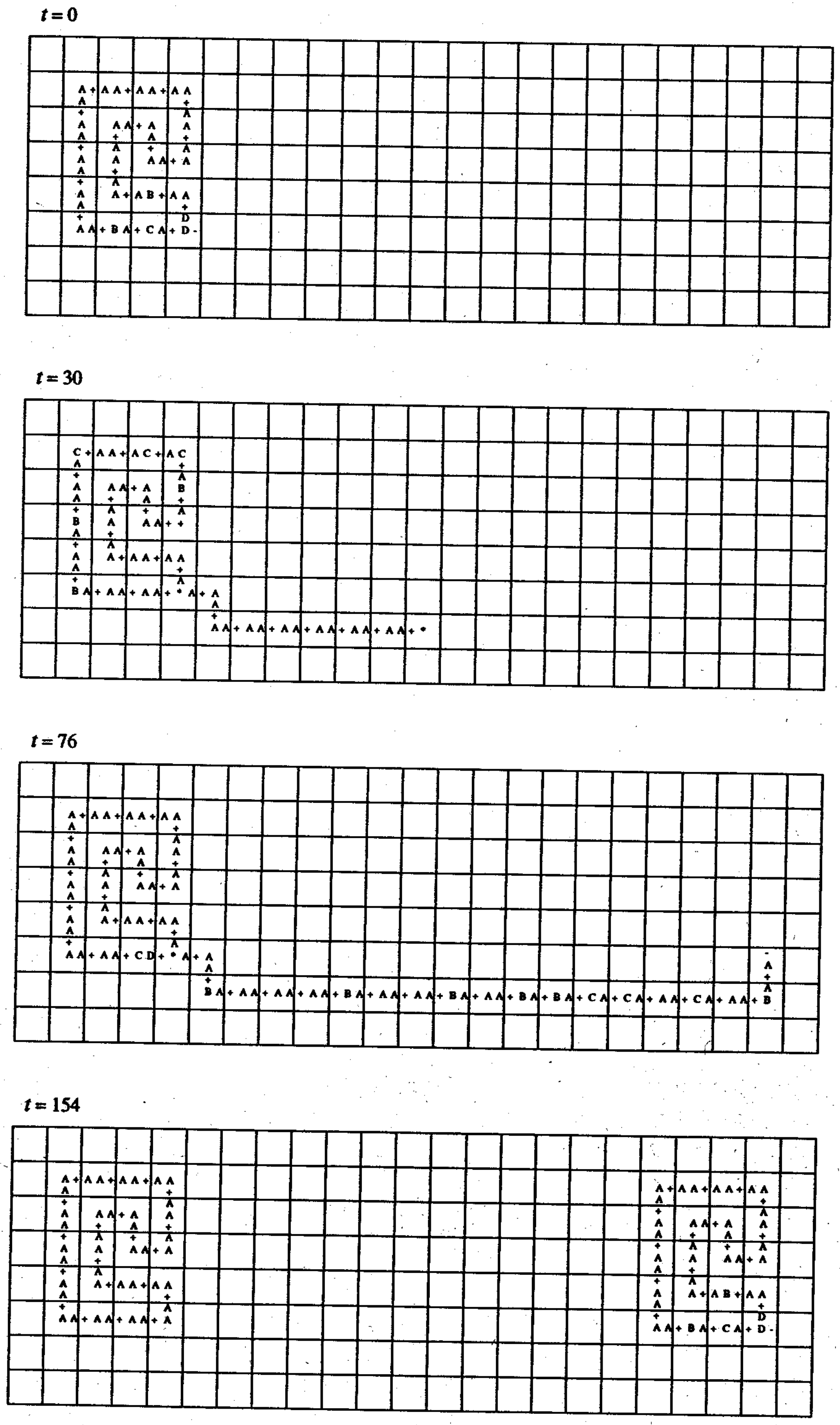


Figure 5.9: Self-reproducing process of a Loop of modified SR_8

5.2 Self-Reproduction in a Three-Dimensional RPCA

5.2.1 An Extension of SR_8 into a Rotation-Symmeric Three-Dimensional RPCA

In this section, we extend SR_8 into a three-dimensional RPCA.

A two-dimensional 5-neighbor PCA can be embedded directly into a three-dimensional 7-neighbor PCA (Fig.5.10). But due to the rotation-symmetric condition of SR_8, the Worm cannot know the directions of right, left, up and down. In three-dimensional rotation-symmetric CA, up to 24 rotated rules are regarded as the same rule. So we introduce another glue state

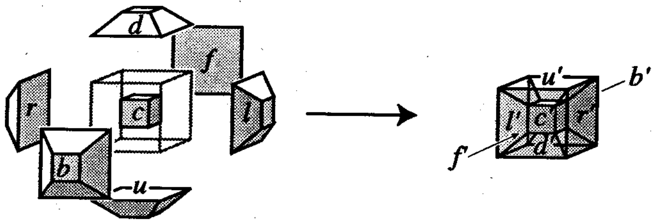


Figure 5.10: Domain and rage of local function in 3D 7-neighbor PCA

'=' for SR_8 and combine two Worms whose command sequences are complementaly placed as presented in table 5.3. An example of the Worm of width 2 is depicted in Fig.5.11. Each Worms can determin the direction by a glue symbol '=' between itself and the opponent.

Table 5.3: Commands for width 2 shaped worm

Command				Operations
wire 1		wire 2		
First Signal	Second Signal	First Signal	Second Signal	
A	A	A	A	Advance the head forward
A	B	A	C	Advance the head leftward
A	C	A	B	Advance the head rightward
B	A	B	A	Branch the wire in three ways
B	B	B	C	Branch the wire in two ways (leftward)
B	C	B	B	Branch the wire in two ways (lrightward)

This ribbon of width 2 shaped Worm in a three-dimensional rotation-symmetric RCA has completly the same behavior as that of SR_8. But this Worm cannot construct three-dimensional

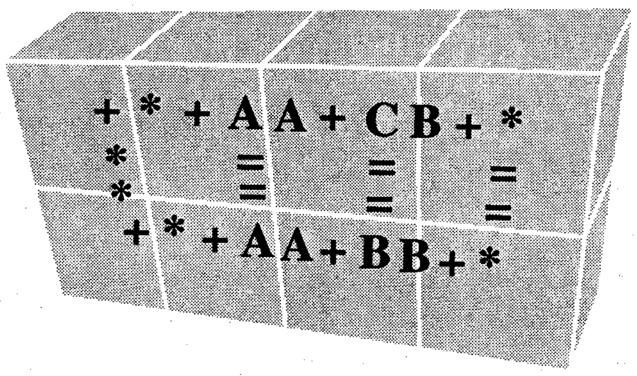


Figure 5.11: A width 2 shaped Worm

shape and only crawls in a plane. In the next section, we employ ribbon of width 3 shaped Worms and show how they perform three-dimensional self-reproduction.

5.2.2 Three-Dimensional Self-Reproducing RPCA (SR_9)

The three-dimensional self-reproducing RPCA "SR_9" is defined by

$$SR_9 = (\mathbf{Z}^3, (C, U, R, D, L, F, B), g, (\#, \#, \#, \#, \#, \#, \#)),$$
$$C = U = R = D = L = F = B = \{\#, *, +, -, =, \text{A,B,C,D}\}.$$
Local rules are shown in Appendix. C.

Although SR_9 has 6886 rules, if rotated rules are regarded as equivalent, it become only 338 rules.

To construct 'true' three-dimensional structures, A Worm in SR_9 can twist its head in ± 90 degrees (Fig.5.12). As far as using SR_8 command sequences in rotation-symmetric spaces, the length of path should be kept equal and this width 2 raddar approach in the previous section is impossible. So we add a center wire and Fig.5.13 is a simple Worm in SR_9.

SR_9 has two more commands for twisting heads. 'A'-commands are extended for twisting heads and table 5.3 shows the command set.

When both worms has the same sequence 'AB AA AC' (or 'AC AA AB'), its head is twisted leftward (rightward). Using twisting commands, complex three-dimensional Worms and Loops such as Fig.5.14 are available. Although the existence of twisting commands in SR_9, its self-reproducing mechanism is completly the same as that of SR_8.

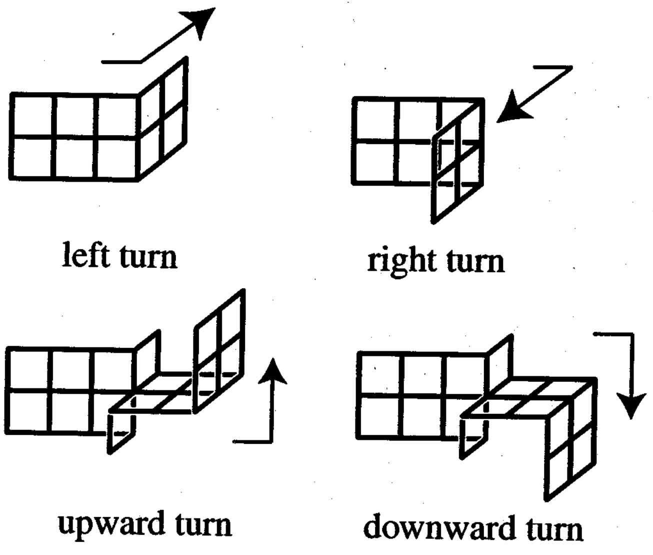


Figure 5.12: Four kind of turns in SR_9

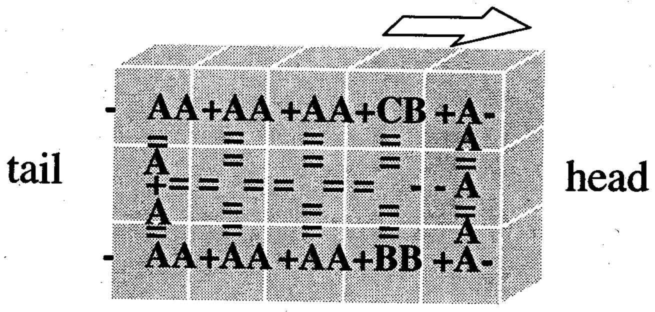


Figure 5.13: A simple Worm in SR_9

5.2.3 Controlling the Position of Daughter Loops in SR_9

When extending SR_8 to SR_9, we use the modified version of SR_8 discussed in section 5.1.6. So Loop positioning commands can also be inserted freely in SR_9. And this modification has an important meaning in the three-dimensional case because it makes possible to generate different topological shapes. Fig.5.15 is a chain formed from a single Loop. This shape-construction technique can be possible by the positioning a daughter Loop with a specific command sequences in the mother Loop.

The self-reproducing processes are hard to describe on a paper. They can be seen as Quick-Time Movies at the following address via WWW.

SR_8: `http://www.ke.sys.hiroshima-u.ac.jp/projects/rca/sr/`

SR_9: `http://kelp.ke.sys.hiroshima-u.ac.jp/projects/rca/sr3d/`

Table 5.4: 'A'-Commands for width 3 shaped worm

Command				Operations
wire 1		wire 3		
First Signal	Second Signal	First Signal	Second Signal	
A	A	A	A	Advance the head forward
A	B	A	C	Advance the head leftward
A	C	A	B	Advance the head rightward
A	B	A	B	Start rotating (leftward)
A	C	A	C	Start rotating (rightward)

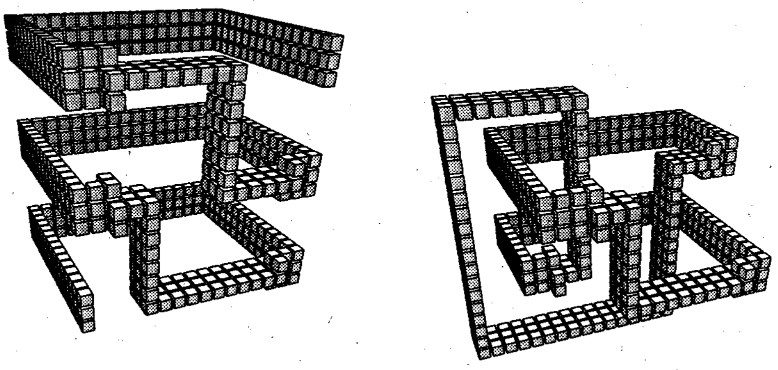


Figure 5.14: Complex Worm and Loop in SR_9

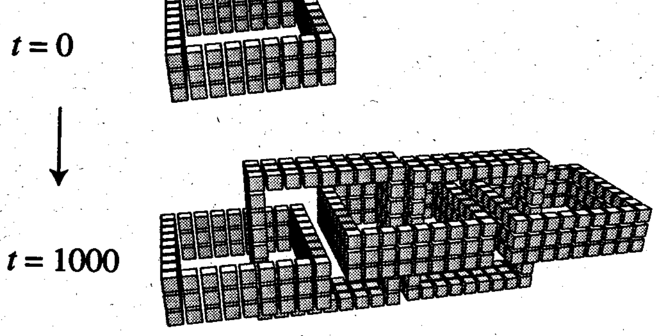


Figure 5.15: A chain formed from a single Loop in SR_9

Chapter 6

Conclusion

In this paper, we have constructed the following cellular models.

- A triangular computation-universal RCAs with small number of states
- Several solutions to the Firing Squad Synchronization Problem to show the abilities of reversible and conservative CAs
- Simple two- and three-dimensional self-reproducing RCAs

In chapter 3, we have constructed an 8-state triangular RPCA which has computation universality. This is the smallest state two-dimensional RPCA with bit-conserving and rotation symmetric local functions. Open problems are as follows.

- To find other 8-state universal RPCA models
- To find universal RPCA models with small number of states

We conjecture that by using partitioning, 6 or 4 state RPCAs can be possible. These RCAs have two-neighbors, and if they are embedded in two or higher dimensional cellular space, they do not have uniform neighbor.

In chapter 4, we defined the Firing Squad Synchronization Problem for reversible and conservative CA and constructed reversible or conservative solution based on Minsky's $3n$-time one. Moreover, we also constructed $2n+2$-time reversible solution based on Goto and Mazoyer's minimal time algorithm. But it is not a minimal time solution. Even if under the constraint of PCA, $2n$-time solution will be possible, i.e., to find

- A reversible solution with minimal number of firing states
- A minimal firing time reversible solution
- A reversible solution with a minimum number of states
- A reversible and conservative solution

are remaining open.

In chapter 5, we constructed simple two- and three-dimensional self-reproducing RCAs by self-inspection method. There are many problems remain in this field.

- To describe the shape constructing ability of SR_9
- To design self-reproducing objects which have some functions in an RCA
- To construct a self-reproducing RCA of von Neumann's sense

The shape constructing ability of SR_9 is not clear now. It might have computation-universality or perform some other interesting functions. But it has not so strong ability as von Neumann's universal constructor. So it is also an open problem to construct a "true" construction-universal self-reproducing RCA.

Bibliography

[1] Adachi, T.: *A Time 2n Reversible Solution to the Firing Squad Synchronization Problem*, Master thesis, Hiroshima university, (1996).

[2] L.Adleman: "Molecular Computation of Solutions to Combinational Problems", *Science,* **266**, (1994), 1021–1024.

[3] Albert, J., and Chulik II, K.: A simple universal cellular automata and its one-way and totalistic version, *Complex Systems*, **1**, (1987), 1–16.

[4] Amoroso, S. and Patt, Y. N.: Decision Procedures for Surjectivity and Injectivity of Parallel Maps for Tessellation Structures, *Journal of Computer and System Sciences*, **6**, 448–464, (1972).

[5] Athas, W.C. and Svensson, L. "J.": Reversible Logic Issues in Adiabatic CMOS *Physcomp '94*, (1994), 111–118.

[6] Baker, G. H.: NREVERSAL of Fortune: The Thermodynamics of Garbage Collection, *Proc. Int'l. Workshop on Memory Mgmt.*, St Malo, France, (1992).

[7] Baker, G. H.: Lively Linear Lisp 'Look Ma, No Garbage!', *ACM Sigplan Notices*, **27**, 8, (1992), 89–98.

[8] Baker, G. H.: Thermodynamics and Garbage Collection *ACM Sigplan Notices*, **29**, 4, (1994), 58–63.

[9] Balzer, R.: An 8-states minimal time solution to the firing squad synchronization problem, *Inform. and Control*, **10**, (1967), 22–42.

[10] Bennett, C.H.: Logical reversibility of computation, *IBM J. Res. Dev.*, **17**, 6, (1973), 525–532.

[11] Bennett, C.H.: The thermodynamics of computation, *Int. J. Theoretical Physics*, **21**, 12, (1982), 905–940.

[12] Bennett, C.H.: Notes on the history of reversible computation, *IBM J. Res. Dev.*, **32**, 1, (1988), 16–23.

[13] Berlekamp, E., Conway, J., Guy, R.: *Winning Ways for Your Mathematical Plays*, Vol. 2, Academic Press, New York (1982).

[14] J. Byl, Self-reproduction in small cellular automata, *Physica D*, **34**, (1989), 295–299.

[15] Clementi, A., Mentrasti, P., Pierini, P.: Some Results on Invertible Cellular Automata *Physcomp '94*, 143–150, (1994).

[16] Codd, E.F.: *Cellular Automata*, Academic Press, New York (1968).

[17] Deutsch, D., Quantum theory, the Church-Turing principle and the universal quantum computer, *Proc.R.Soc.Lond.A*, **400**, (1985), 97–117.

[18] Dooren, G, D. ed.: *Lattice gas methods for partial diferrential equations*, Addison-Wesley, (1987).

[19] Duland-Lose, J.: About the Universality of the Billiard ball model, *Proc. of the Second Colloquium on Universal Machines and Computations*, Volume II (Metz), (1998), 117–132.

[20] Feynman, R.P., Simulating physics with computers, *Int.J.Theoret.Phys.*, **21**, (1982), 467–488.

[21] Fredkin, E., Toffoli, T.: Conservative logic, Int. J. Theoretical Physics, **21**, 3/4, (1982), 219–253.

[22] Fredkin, E.: Digital mechanics: An informational process based on reversible universal CA, *Physica*, **45D**, (1990), 254.

[23] Gardner, M.: MATHEMATICAL GAMES: The fantasic combinations of John Conway's new solitare game "Life", *Scientific American*, October, (1970), 120–123.

[24] Gardner, M.: MATHEMATICAL GAMES: On cellular automata, self-reproduction, the Garden of Eden and the game "Life", *Scientific American*, Feburary, (1971), 112–117.

[25] Garzon, M.: *Models of Massive Parallelizm, Analysis of Cellular Automata and Neural Networks*, Springer, (1995).

[26] Goles, E., Sand pile automata, *Ann. Inst. Henri Poincaré*, **56**, (1992), 75–90.

[27] Goles, E., and Margenstern, M., Sand pile as a universal computer, *Int. J. Modern Physics C*, **7**, (1996), 113–122.

[28] Goto E: A minimal time solution to the firing squad synchronization problem, *Course notes for applied mathematics*, Harvard University, (1962).

[29] 後藤英一: 有限オートマトンと一斉射撃の問題, 情報科学への道 (北川敏男編) 共立出版, (1966), 68–77, (in Japanese).

[30] Gregorio, Di S. and Trautteur G.: On Reversibility in Cellular Automata, *Journal of Computer and Systems Sciences*, **11**, (1975) 382–391.

[31] Hori, T.: *Expanding the self-reproduction model SR_8 in a 2-dimensional reversible PCA into 3-dimensional one*, Master thesis, Hiroshima university, (1997).

[32] Ibáñez, J., Anabitarte, D., Azpeitia, I., Barrera, O., Barrutieta, A., Blanco, H., and Echarte, F., Self-inspection based reproduction in cellular automata, in *Advances in Artificial Life* (eds. F. Moran et al.), LNAI-929, Springer-Verlag, (1995), 564–576.

[33] Imai, K., Morita, K.: Firing Squad Synchronization Problem in One-dimensional Reversible Cellular Automata, *Proc. of LA symposium '93 Winter*, (1994), 66–72, (in Japanese).

[34] Imai, K., Morita, K.: Faster solutions to Firing Squad Synchronization Problem in One-dimensional Reversible Cellular Automata, *Proc. of LA symposium '95 Winter*, (1996), (in Japanese).

[35] Imai, K., Morita, K.: Firing Squad Synchronization Problem in Reversible Cellular Automata, *Theoretical Computer Science*, **165**, (1996), 475–482.

[36] Imai, K., Adachi, T., Furusaka, S. and Morita, K., Firing squad synchronization problem in one and two dimensional reversible cellular automaton, *Proc. of Cellular Automata Workshop '96*, Gießen, (1996), 38–40.

[37] Imai, K., and Morita, K., A computation-universal two-dimensional 8-state triangular reversible cellular automaton, *Proc. of the Second Colloquium on Universal Machines and Computations*, Volume II (Metz), (1998), 90–99.

[38] Imai, K., Morita, K.: A computation-universal two-dimensional 8-state triangular reversible cellular automaton, *Theoretical Computer Science*, (to appear).

[39] Imai, K., Morita, K., and Sako, K.: Firing Squad Synchronization Problem in Number-Conserving Cellular Automata, *Proc. of AUTOMATA'98*, Santiago, (1998).

[40] Kari, J.: Reversibility and Surjectivity Problems of Cellular Automata *Journal of Computer and Systems Sciences*, **48**, (1994), 149–182.

[41] Kari, J.: Representation of Reversible Cellular Automata with Block Permutations, *Math. Systems Theory*, **29**, (1996), 47–61.

[42] Kobayashi, K.: The firing squad synchronization problem for two-dimensional arrays, *Inform. and Control*, **34**, (1977), 177–197.

[43] Kobayashi, K.: On the minimal firing time of the firing squad synchronization problem for polyautomata networks, *Theoretical Computer Science*, **7**, (1978), 149–167.

[44] Langton, C, G.: Self-Reproduction in Cellular Automata, Physica, **10D**, 1/2, (1984), 135–144.

[45] Lindgren, K., Nordahl, M.G.: Universal computation in simple one-dimensional cellular automata *Complex Systems*, 4, (1990), 299–318.

[46] Lohn, J.: Automated Discovery of Self-Replicating Structures in Cellular Space Automata Models, Technical Report UMIACS-TR-96-60 (CS-TR-3677), (1996).

[47] Margolus, N.: Physics-like models of computation, *Physica*, **10D**, 1/2, (1984), 81–95.

[48] Mazoyer, J.: A six-state minimal time solution to the firing squad synchronization problem, *Theoret. Comput. Sci.*, **50**, (1987), 183–238.

[49] Mazoyer, J.: On optimal solutions to the firing squad synchronization problem, *Theoret. Comput. Sci.*, **168**, (1996), 367–404.

[50] Mazoyer, J.: A minimal-time solution to the FSSP without recursive call to itself, *Draft*,(1996).

[51] Minsky, M.: *Computation: Finite and infinite machines*, Prentice Hall, Englewood Cliffs, NJ, (1967).

[52] Moore, E.F.: The firing squad synchronization problem, in *Sequential machines* (ed. E. F. Moore) Addison-Wesley, Reading MA, (1964), 213–214.

[53] Moore F. R. and Langdon G. G.: A Generalized Firing Squad Problem *Information and Control*, **12**, (1968), 212–220.

[54] Morita, K., Shirasaki, A., and Gono, Y.: A 1-tape 2-symbol reversible Turing machine, *Trans. IEICE Japan*, **E72**, 3, (1989), 223–228.

[55] Morita, K., and Harao, M.: Computation universality of one-dimensional reversible (injective) cellular automata, *Trans. IEICE Japan*, **E72**, 6, (1989) 758–762.

[56] Morita, K., Ueno, S: Computation-Universal Models of Two-Dimensional 16-state Reversible Cellular Automata, IEICE Trans. Inf. & Syst., **E75-D**, 1, (1992), 141–147.

[57] Morita, K.: Computation universality of one-dimensional one-way reversible cellular automata, *Inf. Process. Lett.*, **42**, 6, (1992), 325–329.

[58] Morita, K.: Any irreversible cellular automaton can be simulated by a reversible one having the same dimension, Tech. Rep. IEICE Japan, COMP92–45, (1992).

[59] Morita, K.: Reversible Simulation of One-dimensional Irreversible Cellular Automata, *Theoretical Computer Science*, **148**, (1995), 157–163.

[60] Morita, K.: Universality of a reversible two-counter machine, *Theoretical Computer Science*, **168**, (1996), 303–320.

[61] Morita, K., and Imai, K., Self-reproduction in a reversible cellular space, *Theoret.Comput.Sci.*, **168**, 337–366 (1996).
`http://kepi.ke.sys.hiroshima-u.ac.jp/projects/rca/sr/`

[62] Morita, K., and Imai, K., A simple self-reproducing cellular automaton with shape-encoding mechanism, *Proc. ALIFE V*, Nara, (1996).

[63] Morita, K., and Imai, K., Logical universality and self-reproduction in reversible cellular automata, *Evolvable Systems: From Biology to Hardware* (ICES96), LNCS1259, Springer, (1997), 152–166.

[64] Morita, K., Margenstern, M., and Imai, K., Universality of reversible hexagonal cellular automata, MFCS'98 Workshop on Frontiers between Decidability and Undecidability, Brno, (1998).

[65] Morita, K., and Imai, K., Number-Conserving Reversible Cellular Automata and Their Computation-Universality, MFCS'98 Workshop on Cellular Automata, Brno, (1998).

[66] von Neumann, J.: *Theory of Self-reproducing Automata* (ed. A.W.Burks), The University of Illinois Press, Urbana, (1966).

[67] Nishitani, Y. and Honda, N.: The firing squad synchronization problem for graphs, *Theoretical Computer Science*, **14**, (1981), 39–91.

[68] Poundstone, W.: *The Recursive Universe*, International Creative Management, (1985).

[69] Priese, L.: On a simple combinatrial structure sufficient for syblying nontrivial self-reproduction, *Journal of Cybernetics*, **6**, (1976) 101–137.

[70] J.A. Reggia, S.L. Armentrout, H.H. Chou, and Y. Peng, Simple Systems that exhibit self-directed replication, *Science*, **259**, (1993), 1282–1287.

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

[72] Rosenstiehl, P., Fiksel, J. R., Holliger, A.: Intelligent graphs: networks of finite automata capable of solving graph problems, in *Graph Theory and Computing* (ed. R. C. Read) Academic Press, New York, (1972), 219–265.

[73] Serizawa, T.: 3-state Neumann Neghbor Cellular Automata Capable of Constructing Self-Reproducing Machine, *Trans. IEICE Japan*, **J69-D**, 5, (1986), 653–660.

[74] Shikano, T.: *Synchronization problems in reversible cellular automata*, Master thesis, Yamagata university, (1993).

[75] Shinahr, I.: Two- and Three-Dimensional Firing-Squad Synchronization Problems, *Information and Control*, **24**, (1974), 163–180.

[76] Shor, P.: Algorithms for Quantum Computation: Discrete Log and Factoring, *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, 35,(1994), 1-14.

[77] Sipper, M.: Evolution of Parallel Cellular Machines, LNCS1194, Springer, (1997).

[78] Smith III, A.R.: Simple computation-universal cellular spaces *Journal of ACM*, **18**, (1971), 339–353.

[79] Toffoli, T.: Computation and Construction Universality of Reversible Cellular Automata, *Journal of Computer and System Sciences*, **15**, (1977), 213–231.

[80] Toffoli, T., Margolus, M.: *Cellular Automata Machines*, MIT Press, London, (1987).

[81] Toffoli, T., and Margolus, N.: Invertible cellular automata: a review, *Physica*, **45D**, 1/3, (1990), 229–253.

[82] Torre, S., Napoli, M., and Parente, M.: Synchronization of One-way Connected Processors *Complex Systems*, **10**, 239-255, (1996).

[83] Umeo, H.: A Note on Firing Squad Synchronization Algorithms, *Proc. of Cellular Automata Workshop '96*, Gießen, (1996).

[84] Waksman, A.: An optimum solution to the firing squad synchronization problem, *Inform. and Control*, **9**, (1966), 66–78.

[85] Watrous, J., On one-dimensional quantum cellular automata, *Proc. 36th IEEE Symposium on Foundations of Computer Science*, (1995), 528–537.

[86] Wolfram, S.: Universality and complexity in cellular automata *Physica*, **10D**, (1984), 1–35.

[87] Wolfram, S. ed.: *Theory and Applications of Cellular Automata*, World Scientific, Singapore, (1986).

[88] J.B.Yunès: Seven–state solutions to the Firing Squad Synchronization Problem, *Theoret. Comput. Sci.*, **127**,(1994), 313–332.

Appendix A

Transition rules for $2n + 2$-time solution

φ_0 :

```
#0(“0”,“#”,“0”) → #0(“0”,“#”,“0”)
#0(“0”,“#”,“!”) → #0(“0”,“#”,“!”)
#0(“0”,“#”,“+”) → #0(“0”,“#”,“+”)
#0(“0”,“#”,“*”) → #0(“0”,“#”,“*”)
#0(“0”,“#”,“-”) → #0(“0”,“#”,“-”)
#0(“0”,“#”,“””) → #0(“0”,“#”,“””)
#0(“0”,“#”,“@”) → #0(“0”,“#”,“@”)
#0(“0”,“#”,“{”) → #0(“0”,“#”,“{”)
#0(“@”,“#”,“0”) → #0(“@”,“#”,“0”)
#0(“-”,“#”,“0”) → #0(“-”,“#”,“0”)
#0(“[”,“#”,“0”) → #0(“[”,“#”,“0”)
#0(“””,“#”,“0”) → #0(“””,“#”,“0”)
#0(“0”,“Gk”,“0”) → #0(“0”,“Gk”,“0”)
#0(“+”,“0”,“0”) → #0(“0”,“0”,“+”)
#0(“0”,“0”,“+”) → #0(“+”,“0”,“0”)
#0(“@”,“0”,“0”) → #0(“0”,“0”,“@”)
#0(“0”,“0”,“@”) → #0(“@”,“0”,“0”)
#0(“+”,“G”,“0”) → #0(“+”,“Gk”,“0”)
#0(“+”,“Gk”,“0”) → #0(“0”,“G”,“+”)
#0(“0”,“G”,“+”) → #0(“+”,“G”,“0”)
#0(“+”,“M”,“0”) → #0(“0”,“M”,“+”)
#0(“[”,“M”,“+”) → #0(“+”,“M”,“[”)
#0(“0”,“M”,“+”) → #0(“+”,“M”,“0”)
#0(“+”,“E”,“0”) → #0(“0”,“E”,“+”)
#0(“0”,“E”,“+”) → #0(“+”,“E”,“0”)
#0(“[”,“E”,“+”) → #0(“+”,“E”,“[”)
#0(“+”,“K”,“0”) → #0(“0”,“K”,“+”)
#0(“0”,“K”,“+”) → #0(“+”,“K”,“0”)
#0(“+”,“A”,“0”) → #0(“0”,“A”,“+”)
#0(“+”,“A^ ”,“0”) → #0(“0”,“A^ ”,“+”)
#0(“+”,“Z”,“0”) → #0(“0”,“Z”,“+”)
#0(“+”,“B”,“0”) → #0(“0”,“B”,“+”)
#0(“+”,“B^ ”,“0”) → #0(“0”,“B^ ”,“+”)
#0(“+”,“Y”,“0”) → #0(“0”,“Y”,“+”)
#0(“+”,“[”,“0”) → #0(“0”,“[”,“+”)
#0(“+”,“0”,“*”) → #0(“*”,“0”,“+”)
#0(“+”,“G”,“*”) → #0(“!”,“Gk”,“0”)
#0(“!”,“Gk”,“0”) → #0(“0”,“G”,“!”)
#0(“!”,“G”,“0”) → #0(“+”,“Gk”,“$”)
#0(“!”,“B”,“0”) → #0(“*”,“B”,“+”)
#0(“!”,“Y”,“0”) → #0(“*”,“Y”,“+”)
#0(“!”,“0”,“0”) → #0(“0”,“c”,“+”)
#0(“*”,“0”,“+”) → #0(“+”,“c”,“0”)
#0(“’”,“0”,“+”) → #0(“+”,“c”,“^ ”)
#0(“0”,“c”,“+”) → #0(“+”,“d”,“0”)
#0(“+”,“c”,“0”) → #0(“0”,“d”,“+”)
#0(“+”,“c”,“-”) → #0(“-”,“d”,“+”)
#0(“0”,“d”,“+”) → #0(“+”,“e”,“0”)
#0(“+”,“d”,“0”) → #0(“0”,“e”,“+”)
#0(“+”,“d”,“-”) → #0(“-”,“e”,“+”)
#0(“0”,“e”,“+”) → #0(“+”,“0”,“*”)
```

```
#0(“0”,“Mc”,“+”) → #0(“+”,“Md”,“0”)
#0(“+”,“Mc”,“0”) → #0(“0”,“Md”,“+”)
#0(“+”,“Mc”,“_”) → #0(“-”,“Md”,“+”)
#0(“0”,“Md”,“+”) → #0(“+”,“Me”,“0”)
#0(“+”,“Md”,“0”) → #0(“0”,“Me”,“+”)
#0(“+”,“Md”,“_”) → #0(“-”,“Me”,“+”)
#0(“0”,“Me”,“+”) → #0(“+”,“M”,“’”)
#0(“+”,“b”,“0”) → #0(“+”,“E”,“(”)
#0(“+”,“b”,“-”) → #0(“_”,“C”,“@”)
#0(“+”,“0”,“/”) → #0(“-”,“B”,“+”)
#0(“+”,“0”,“-”) → #0(“-”,“0”,“+”)
#0(“+”,“0”,“_”) → #0(“_”,“0”,“+”)
#0(“^ ”,“0”,“+”) → #0(“+”,“0”,“^ ”)
#0(“+”,“G”,“-”) → #0(“@”,“Gk”,“0”)
#0(“0”,“Gk”,“-”) → #0(“-”,“Gk”,“0”)
#0(“@”,“Gk”,“0”) → #0(“0”,“G”,“@”)
#0(“@”,“Gk”,“*”) → #0(“*”,“G”,“@”)
#0(“@”,“G”,“0”) → #0(“+”,“Gk”,“-”)
#0(“+”,“Gk”,“-”) → #0(“-”,“G”,“+”)
#0(“[”,“Gk”,“-”) → #0(“-”,“Gk”,“[”)
#0(“[”,“Gk”,“0”) → #0(“0”,“Gk”,“[”)
#0(“+”,“Gk”,“{”) → #0(“{”,“G”,“+”)
#0(“+”,“M”,“_”) → #0(“-”,“M”,“+”)
#0(“+”,“E”,“-”) → #0(“_”,“E”,“+”)
#0(“+”,“K”,“-”) → #0(“_”,“K”,“+”)
#0(“+”,“M”,“*”) → #0(“*”,“M”,“+”)
#0(“+”,“M”,“}”) → #0(“{”,“M”,“+”)
#0(“+”,“M¯ ”,“0”) → #0(“0”,“M¯ ”,“+”)
#0(“!”,“G”,“””) → #0(“””,“Gk”,“$”)
#0(“1”,“0”,“0”) → #0(“0”,“0”,“1”)
#0(“0”,“A”,“+”) → #0(“+”,“A”,“0”)
#0(“0”,“M¯ ”,“+”) → #0(“+”,“M¯ ”,“0”)
#0(“0”,“C”,“+”) → #0(“+”,“C”,“0”)
#0(“-”,“M”,“+”) → #0(“+”,“M”,“-”)
#0(“@”,“M”,“0”) → #0(“0”,“M”,“@”)
#0(“@”,“M”,“*”) → #0(“*”,“M”,“@”)
#0(“@”,“E”,“0”) → #0(“0”,“E”,“@”)
#0(“@”,“[”,“0”) → #0(“0”,“[”,“@”)
#0(“0”,“0”,“*”) → #0(“*”,“0”,“0”)
#0(“0”,“G”,“*”) → #0(“*”,“G”,“0”)
#0(“*”,“G”,“0”) → #0(“0”,“G”,“*”)
#0(“-”,“G”,“*”) → #0(“*”,“G”,“-”)
#0(“*”,“G”,“-”) → #0(“-”,“G”,“*”)
#0(“0”,“E”,“*”) → #0(“*”,“E”,“0”)
#0(“0”,“K”,“*”) → #0(“*”,“K”,“0”)
#0(“0”,“M”,“*”) → #0(“*”,“M”,“0”)
#0(“0”,“K”,“{”) → #0(“}”,“K”,“0”)
#0(“0”,“G”,“{”) → #0(“{”,“G”,“0”)
#0(“{”,“G”,“0”) → #0(“0”,“G”,“{”)
#0(“{”,“M”,“0”) → #0(“0”,“M”,“{”)
#0(“{”,“K”,“0”) → #0(“0”,“K”,“{”)
```

```
#0(“*”,“0”,“0”) → #0(“0”,“c”,“0”)
#0(“*”,“0”,“-”) → #0(“-”,“c”,“0”)
#0(“*”,“0”,“_”) → #0(“_”,“c”,“0”)
#0(“0”,“c”,“0”) → #0(“0”,“d”,“0”)
#0(“0”,“c”,“-”) → #0(“-”,“d”,“0”)
#0(“0”,“c”,“_”) → #0(“_”,“d”,“0”)
#0(“0”,“d”,“0”) → #0(“0”,“e”,“0”)
#0(“0”,“d”,“-”) → #0(“-”,“e”,“0”)
#0(“0”,“d”,“_”) → #0(“_”,“e”,“0”)
#0(“0”,“e”,“0”) → #0(“0”,“0”,“*”)
#0(“0”,“e”,“-”) → #0(“-”,“0”,“*”)
#0(“0”,“e”,“_”) → #0(“_”,“0”,“*”)
#0(“*”,“M”,“0”) → #0(“0”,“Mc”,“0”)
#0(“0”,“Mc”,“0”) → #0(“0”,“Md”,“0”)
#0(“0”,“Md”,“0”) → #0(“0”,“Me”,“0”)
#0(“0”,“Me”,“0”) → #0(“0”,“M”,“’”)
#0(“*”,“E”,“0”) → #0(“0”,“Ec”,“0”)
#0(“0”,“Ec”,“0”) → #0(“0”,“Ed”,“0”)
#0(“0”,“Ed”,“0”) → #0(“0”,“Ee”,“0”)
#0(“0”,“Ee”,“0”) → #0(“0”,“E”,“$”)
#0(“0”,“Ec”,“-”) → #0(“_”,“Ed”,“0”)
#0(“0”,“Ed”,“-”) → #0(“_”,“Ee”,“0”)
#0(“0”,“Ee”,“-”) → #0(“_”,“E”,“$”)
#0(“0”,“Kc”,“0”) → #0(“0”,“Kd”,“0”)
#0(“0”,“Kd”,“0”) → #0(“0”,“Ke”,“0”)
#0(“0”,“Ke”,“0”) → #0(“0”,“K”,“$”)
#0(“0”,“Ke”,“-”) → #0(“_”,“K”,“$”)
#0(“*”,“A”,“0”) → #0(“*”,“A”,“0”)
#0(“*”,“A^ ”,“0”) → #0(“*”,“A^ ”,“0”)
#0(“*”,“Z”,“0”) → #0(“*”,“Z”,“0”)
#0(“*”,“Z^ ”,“0”) → #0(“*”,“Z^ ”,“0”)
#0(“*”,“C”,“0”) → #0(“*”,“C”,“0”)
#0(“*”,“B”,“0”) → #0(“*”,“B”,“0”)
#0(“*”,“B^ ”,“0”) → #0(“*”,“B^ ”,“0”)
#0(“*”,“Y”,“0”) → #0(“*”,“Y”,“0”)
#0(“*”,“D”,“0”) → #0(“*”,“D”,“0”)
#0(“$”,“0”,“0”) → #0(“0”,“c”,“=”)
#0(“$”,“0”,“-”) → #0(“-”,“c”,“=”)
#0(“$”,“0”,“/”) → #0(“{”,“B”,“/”)
#0(“$”,“b”,“0”) → #0(“0”,“Mc”,“&”)
#0(“’”,“E”,“0”) → #0(“0”,“Kc”,“0”)
#0(“’”,“B”,“0”) → #0(“*”,“B^ ”,“/”)
#0(“$”,“B”,“0”) → #0(“*”,“B”,“/”)
#0(“$”,“Y”,“0”) → #0(“*”,“Y”,“/”)
#0(“’”,“0”,“0”) → #0(“0”,“c”,“^ ”)
#0(“’”,“0”,“-”) → #0(“-”,“c”,“^ ”)
#0(“’”,“0”,“_”) → #0(“_”,“c”,“^ ”)
#0(“0”,“Kc”,“/”) → #0(“_”,“Id”,“0”)
#0(“0”,“Id”,“0”) → #0(“0”,“Ie”,“0”)
#0(“0”,“Ie”,“0”) → #0(“*”,“I”,“0”)
#0(“””,“Gk”,“*”) → #0(“*”,“G”,“””)
```

```
#0(“/”,“0”,“0”) → #0(“0”,“a”,“0”)
#0(“0”,“a”,“0”) → #0(“0”,“b”,“0”)
#0(“0”,“b”,“0”) → #0(“0”,“0”,“/”)
#0(“0”,“b”,“-”) → #0(“-”,“A”,“-”)
#0(“[”,“b”,“0”) → #0(“0”,“0”,“)”)
#0(“=”,“b”,“0”) → #0(“0”,“M”,“&”)
#0(“””,“0”,“0”) → #0(“0”,“a”,“-”)
#0(“&”,“0”,“0”) → #0(“0”,“a”,“,”)
#0(“(”,“0”,“0”) → #0(“0”,“a”,“.”)
#0(“)”,“0”,“0”) → #0(“0”,“a”,“[”)
#0(“/”,“0”,“-”) → #0(“/”,“A˜ ”,“0”)
#0(“&”,“0”,“-”) → #0(“/”,“A-”,“0”)
#0(“(”,“0”,“-”) → #0(“/”,“C-”,“0”)
#0(“””,“0”,“-”) → #0(“””,“G”,“0”)
#0(“””,“Gk”,“0”) → #0(“0”,“G”,“””)
#0(“””,“G”,“””) → #0(“””,“G3”,“0”)
#0(“-”,“0”,“0”) → #0(“0”,“0”,“-”)
#0(“0”,“0”,“-”) → #0(“-”,“0”,“0”)
#0(“0”,“0”,“-”) → #0(“-”,“0”,“0”)
#0(“-”,“0”,“-”) → #0(“-”,“G”,“0”)
#0(“-”,“G”,“0”) → #0(“0”,“G”,“-”)
#0(“˜ ”,“0”,“0”) → #0(“0”,“0”,“˜ ”)
#0(“=”,“0”,“-”) → #0(“-”,“0”,“=”)
#0(“ˆ ”,“0”,“-”) → #0(“-”,“0”,“ˆ ”)
#0(“ˆ ”,“0”,“-”) → #0(“-”,“0”,“ˆ ”)
#0(“,”,“0”,“-”) → #0(“-”,“M”,“0”)
#0(“.”,“0”,“-”) → #0(“-”,“E”,“0”)
```

```
#0(“[”,“0”,“-”) → #0(“-”,“[”,“0”)
#0(“0”,“G”,“-”) → #0(“-”,“G”,“0”)
#0(“[”,“G”,“-”) → #0(“-”,“G”,“[”)
#0(“-”,“M”,“0”) → #0(“0”,“M”,“-”)
#0(“-”,“E”,“0”) → #0(“0”,“E”,“-”)
#0(“ˆ ”,“E”,“-”) → #0(“-”,“E”,“[”)
#0(“0”,“K”,“-”) → #0(“-”,“K”,“0”)
#0(“0”,“A-”,“0”) → #0(“0”,“A”,“-”)
#0(“0”,“A˜ ”,“0”) → #0(“0”,“A”,“˜ ”)
#0(“0”,“Z-”,“0”) → #0(“0”,“Z”,“-”)
#0(“0”,“C-”,“0”) → #0(“0”,“C”,“-”)
#0(“-”,“[”,“0”) → #0(“0”,“[”,“-”)
#0(“[”,“0”,“-”) → #0(“-”,“0”,“[”)
#0(“˜ ”,“M”,“0”) → #0(“0”,“M˜ ”,“-”)
#0(“0”,“M˜ ”,“0”) → #0(“0”,“M˜ ”,“0”)
#0(“0”,“M˜ ”,“-”) → #0(“-”,“M˜ ”,“0”)
#0(“0”,“C”,“-”) → #0(“-”,“C”,“0”)
#0(“@”,“I”,“0”) → #0(“0”,“I”,“@”)
#0(“0”,“G2”,“-”) → #0(“””,“G”,“””)
#0(“=”,“0”,“0”) → #0(“0”,“0”,“=”)
#0(“ˆ ”,“0”,“0”) → #0(“0”,“0”,“ˆ ”)
#0(“0”,“Aˆ ”,“0”) → #0(“0”,“Aˆ ”,“0”)
#0(“=”,“B”,“0”) → #0(“0”,“B”,“/”)
#0(“ˆ ”,“B”,“0”) → #0(“0”,“Bˆ ”,“/”)
#0(“0”,“Bˆ ”,“0”) → #0(“0”,“Bˆ ”,“0”)
#0(“ˆ ”,“D”,“0”) → #0(“0”,“D”,“[”)
#0(“=”,“Y”,“0”) → #0(“0”,“Y”,“/”)
```

```
#0(“0”,“0”,“0”) → #0(“0”,“0”,“0”)
#0(“,”,“0”,“0”) → #0(“0”,“0”,“,”)
#0(“.”,“0”,“0”) → #0(“0”,“0”,“.”)
#0(“[”,“0”,“0”) → #0(“0”,“0”,“[”)
#0(“0”,“G3”,“0”) → #0(“0”,“G2”,“0”)
#0(“0”,“G2”,“0”) → #0(“!”,“G”,“””)
#0(“0”,“G”,“0”) → #0(“0”,“G”,“0”)
#0(“0”,“M”,“0”) → #0(“0”,“M”,“0”)
#0(“0”,“E”,“0”) → #0(“0”,“E”,“0”)
#0(“0”,“K”,“0”) → #0(“0”,“K”,“0”)
#0(“0”,“I”,“0”) → #0(“0”,“I”,“0”)
#0(“0”,“A”,“0”) → #0(“0”,“A”,“0”)
#0(“0”,“Z”,“0”) → #0(“0”,“Z”,“0”)
#0(“0”,“C”,“0”) → #0(“0”,“C”,“0”)
#0(“0”,“B”,“0”) → #0(“0”,“B”,“0”)
#0(“0”,“Y”,“0”) → #0(“0”,“Y”,“0”)
#0(“0”,“D”,“0”) → #0(“0”,“D”,“0”)
#0(“0”,“[”,“0”) → #0(“0”,“[”,“0”)
#0(“[”,“B”,“0”) → #0(“0”,“B”,“[”)
#0(“[”,“A”,“0”) → #0(“0”,“A”,“[”)
#0(“[”,“[”,“0”) → #0(“0”,“[”,“[”)
#0(“[”,“C”,“0”) → #0(“0”,“C”,“[”)
#0(“[”,“E”,“0”) → #0(“0”,“E”,“[”)
#0(“[”,“M”,“0”) → #0(“0”,“M”,“[”)
#0(“[”,“G”,“0”) → #0(“0”,“G”,“[”)
```

φ_1 :

```
#1(“0”,“#”,“0”) → #1(“0”,“#”,“0”)
#1(“*”,“#”,“0”) → #1(“*”,“#”,“0”)
#1(“0”,“#”,“*”) → #1(“0”,“#”,“*”)
#1(“0”,“#”,“+”) → #1(“+”,“#”,“0”)
#1(“+”,“#”,“0”) → #1(“0”,“#”,“+”)
#1(“0”,“0r”,“0”) → #1(“0”,“0r”,“0”)
#1(“0”,“0r”,“*”) → #1(“*”,“t”,“+”)
#1(“0”,“0r”,“+”) → #1(“*”,“u”,“+”)
#1(“*”,“0r”,“*”) → #1(“+”,“er”,“+”)
#1(“*”,“0r”,“+”) → #1(“0”,“wr”,“+”)
#1(“+”,“0r”,“*”) → #1(“0”,“er”,“+”)
#1(“+”,“0r”,“+”) → #1(“+”,“wr”,“+”)
#1(“0”,“0l”,“0”) → #1(“0”,“0l”,“0”)
#1(“*”,“0l”,“0”) → #1(“+”,“t”,“*”)
#1(“+”,“0l”,“0”) → #1(“+”,“u”,“*”)
#1(“*”,“0l”,“*”) → #1(“+”,“el”,“+”)
#1(“+”,“0l”,“*”) → #1(“+”,“wl”,“0”)
#1(“*”,“0l”,“+”) → #1(“+”,“el”,“0”)
#1(“+”,“0l”,“+”) → #1(“+”,“wl”,“+”)
#1(“0”,“t”,“0”) → #1(“0”,“t”,“0”)
```

```
#1(“*”,“t”,“0”) → #1(“0”,“0r”,“*”)
#1(“+”,“t”,“0”) → #1(“0”,“t”,“+”)
#1(“0”,“t”,“*”) → #1(“*”,“0l”,“0”)
#1(“+”,“t”,“*”) → #1(“*”,“0r”,“*”)
#1(“0”,“t”,“+”) → #1(“+”,“t”,“0”)
#1(“*”,“t”,“+”) → #1(“*”,“0l”,“*”)
#1(“+”,“t”,“+”) → #1(“0”,“u”,“0”)
#1(“0”,“u”,“0”) → #1(“0”,“v”,“0”)
#1(“*”,“u”,“0”) → #1(“*”,“0r”,“0”)
#1(“+”,“u”,“0”) → #1(“+”,“0r”,“0”)
#1(“0”,“u”,“*”) → #1(“0”,“0l”,“*”)
#1(“0”,“u”,“+”) → #1(“0”,“0l”,“+”)
#1(“0”,“v”,“0”) → #1(“*”,“u”,“*”)
#1(“*”,“v”,“0”) → #1(“+”,“v”,“0”)
#1(“+”,“v”,“0”) → #1(“+”,“0l”,“0”)
#1(“0”,“v”,“*”) → #1(“0”,“v”,“+”)
#1(“0”,“v”,“+”) → #1(“0”,“0r”,“+”)
#1(“0”,“wr”,“0”) → #1(“0”,“wr”,“0”)
#1(“0”,“wr”,“*”) → #1(“0”,“wr”,“*”)
#1(“0”,“wr”,“+”) → #1(“0”,“f”,“+”)
```

```
#1(“*”,“wr”,“*”) → #1(“*”,“wr”,“*”)
#1(“+”,“wr”,“0”) → #1(“+”,“wr”,“0”)
#1(“+”,“wr”,“*”) → #1(“+”,“wr”,“*”)
#1(“+”,“wr”,“+”) → #1(“*”,“f”,“+”)
#1(“0”,“wl”,“0”) → #1(“0”,“wl”,“0”)
#1(“*”,“wl”,“0”) → #1(“*”,“wl”,“0”)
#1(“+”,“wl”,“0”) → #1(“+”,“f”,“0”)
#1(“*”,“wl”,“*”) → #1(“*”,“wl”,“*”)
#1(“0”,“wl”,“+”) → #1(“0”,“wl”,“+”)
#1(“*”,“wl”,“+”) → #1(“*”,“wl”,“+”)
#1(“+”,“wl”,“+”) → #1(“+”,“f”,“*”)
#1(“0”,“er”,“0”) → #1(“0”,“f”,“0”)
#1(“+”,“er”,“0”) → #1(“*”,“f”,“0”)
#1(“0”,“el”,“0”) → #1(“+”,“f”,“+”)
#1(“0”,“el”,“+”) → #1(“0”,“f”,“*”)
#1(“*”,“wr”,“0”) → #1(“*”,“wr”,“0”)
#1(“0”,“wl”,“*”) → #1(“0”,“wl”,“*”)
```

φ_{01} :

```
#0(“+”,“Mc”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“Md”,“+”)
#0(“+”,“Md”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“Me”,“+”)
#0(“+”,“Md”,“/”), #1(“0”,“0r”,“+”) → #0(“-”,“B”,“+”)
#0(“+”,“M”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“M”,“+”)
#0(“+”,“M”,“{”), #1(“0”,“0r”,“0”) → #0(“{”,“M”,“+”)
#0(“1”,“M”,“0”), #1(“0”,“0l”,“0”) → #0(“0”,“M˜ ”,“@”)
#0(“0”,“Mc”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“Md”,“0”)
#0(“0”,“Md”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“Me”,“0”)
#0(“0”,“Me”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“M”,“’”)
#0(“$”,“b”,“-”), #1(“0”,“0r”,“0”) → #0(“{”,“A”,“-”)
#0(“’”,“A”,“0”), #1(“0”,“0r”,“0”) → #0(“*”,“Aˆ ”,“0”)
#0(“$”,“A”,“0”), #1(“0”,“0r”,“0”) → #0(“*”,“A”,“0”)
#0(“$”,“Z”,“0”), #1(“0”,“0r”,“0”) → #0(“*”,“Z”,“0”)
#0(“$”,“C”,“0”), #1(“0”,“0r”,“0”) → #0(“*”,“C”,“0”)
#0(“’”,“C”,“0”), #1(“0”,“wl”,“0”) → #0(“*”,“C”,“[”)
#0(“$”,“G”,“0”), #1(“*”,“0r”,“*”) → #0(“*”,“G”,“0”)
#0(“=”,“b”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“A”,“-”)
#0(“0”,“b”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“A”,“˜ ”)
#0(“[”,“b”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“Z”,“-”)
#0(“/”,“0”,“-”), #1(“0”,“0r”,“0”) → #0(“/”,“A-”,“0”)
```

```
#0(“&”,“0”,“-”), #1(“0”,“0r”,“0”) → #0(“/”,“A-”,“0”)
#0(“=”,“0”,“/”), #1(“0”,“0r”,“0”) → #0(“-”,“B”,“/”)
#0(“0”,“0”,“/”), #1(“0”,“0r”,“0”) → #0(“-”,“B”,“0”)
#0(“[”,“0”,“/”), #1(“0”,“0r”,“0”) → #0(“-”,“Y”,“0”)
#0(“)”,“0”,“-”), #1(“0”,“0r”,“0”) → #0(“/”,“Z-”,“0”)
#0(“0”,“M”,“/”), #1(“0”,“0r”,“+”) → #0(“-”,“B”,“0”)
#0(“0”,“E”,“/”), #1(“0”,“wl”,“0”) → #0(“-”,“D”,“0”)
#0(“ˆ ”,“E”,“/”), #1(“0”,“wl”,“0”) → #0(“-”,“D”,“[”)
#0(“/”,“A”,“0”), #1(“0”,“0r”,“0”) → #0(“0”,“A”,“0”)
#0(“/”,“Z”,“0”), #1(“0”,“0r”,“0”) → #0(“0”,“Z”,“0”)
#0(“0”,“M”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“M”,“0”)
#0(“0”,“M”,“-”), #1(“0”,“0r”,“0”) → #0(“-”,“M”,“0”)
#0(“0”,“E”,“-”), #1(“0”,“wl”,“0”) → #0(“-”,“E”,“0”)
#0(“0”,“E”,“-”), #1(“0”,“wl”,“*”) → #0(“-”,“E”,“0”)
#0(“ˆ ”,“E”,“0”), #1(“0”,“wl”,“0”) → #0(“0”,“E”,“[”)
#0(“ˆ ”,“A”,“0”), #1(“0”,“0r”,“0”) → #0(“0”,“Aˆ ”,“0”)
#0(“=”,“A”,“0”), #1(“0”,“0r”,“0”) → #0(“0”,“A”,“0”)
#0(“=”,“Z”,“0”), #1(“0”,“0r”,“0”) → #0(“0”,“Z”,“0”)
#0(“ˆ ”,“C”,“0”), #1(“0”,“wl”,“0”) → #0(“0”,“C”,“[”)
```

φ_{10} :

```
#1("0","0r","0"), #0("0","0","-") → #1("0","0l","0")
#1("0","0r","0"), #0("[","0","-") → #1("0","0l","0")
#1("0","0r","0"), #0(",","0","-") → #1("0","0l","0")
#1("0","0r","0"), #0("+","0","-") → #1("0","0l","0")
#1("0","0r","0"), #0("^ ","0","-") → #1("0","0l","0")
#1("0","0r","0"), #0("'","0","-") → #1("0","0l","0")
#1("0","0r","0"), #0("0","0","‘") → #1("0","0l","0")
#1("0","0r","0"), #0("*","0","-") → #1("0","0l","0")
#1("0","0r","0"), #0("0","c","-") → #1("0","0l","0")
#1("0","0r","0"), #0("0","d","-") → #1("0","0l","0")
#1("0","0r","0"), #0("0","e","-") → #1("0","0l","0")
#1("0","0r","0"), #0("~ ","0","0") → #1("0","0l","0")
#1("0","0r","0"), #0("1","0","0") → #1("0","0l","0")
#1("0","0r","0"), #0(".","0","-") → #1("0","wr","0")
#1("0","0r","0"), #0("(","0","-") → #1("0","wr","0")
#1("0","0r","0"), #0("-","0","-") → #1("0","wl","0")
#1("0","0r","0"), #0("+","b","-") → #1("0","wl","0")
#1("0","0r","0"), #0("+","b","0") → #1("0","wl","0")
#1("0","0r","0"), #0("=","A","0") → #1("*","0r","*")
#1("0","0r","0"), #0("^ ","A","0") → #1("*","0r","*")
#1("0","0r","0"), #0("$","A","0") → #1("*","0r","*")
#1("0","0r","0"), #0("'","A","0") → #1("*","0r","*")
#1("0","0r","0"), #0("$","C","0") → #1("*","0r","*")
#1("0","0r","0"), #0("$","Z","0") → #1("*","0r","*")
#1("0","0r","0"), #0("=","Z","0") → #1("*","0r","*")
#1("0","0r","0"), #0("0","M","-") → #1("*","0r","*")

#1("0","0r","0"), #0("+","M","-") → #1("*","0r","*")
#1("0","0r","0"), #0("0","M","}") → #1("*","0r","*")
#1("0","0r","0"), #0("+","M","}") → #1("*","0r","*")
#1("0","0l","0"), #0("-","M","0") → #1("*","0r","*")
#1("0","0l","0"), #0("@","M","0") → #1("*","0r","*")
#1("0","0r","0"), #0("=","b","-") → #1("*","0r","*")
#1("0","0r","0"), #0("$","b","-") → #1("*","0r","*")
#1("0","0r","0"), #0(""","0","-") → #1("*","0r","*")
#1("0","0r","0"), #0("&","0","-") → #1("+","v","0")
#1("0","0r","0"), #0("/","A","0") → #1("+","v","0")
#1("0","0r","0"), #0("/","Z","0") → #1("+","v","0")
#1("0","0r","0"), #0("0","M","-") → #1("*","wr","0")
#1("0","0r","0"), #0("+","M","-") → #1("*","wr","0")
#1("0","0r","0"), #0("0","M","{") → #1("*","wr","*")
#1("0","0r","0"), #0("+","M","{") → #1("*","wr","*")
#1("0","0r","0"), #0("0","M","@") → #1("*","wr","0")
#1("0","0r","0"), #0("0","Mc","-") → #1("*","wr","0")
#1("0","0r","0"), #0("+","Mc","-") → #1("*","wr","0")
#1("0","0r","0"), #0("0","Me","-") → #1("*","wr","0")
#1("0","0r","0"), #0("0","Me","@") → #1("*","wr","0")
#1("0","0l","0"), #0("~ ","M","0") → #1("0","wl","*")
#1("0","0l","0"), #0("1","M","0") → #1("0","wl","*")
#1("0","0r","0"), #0("/","0","-") → #1("0","0r","0")
#1("0","wr","0"), #0(""","G",""") → #1("-","f","-")
```

Appendix B

Transition rules for SR_8

The set of 765 rules of SR_8. (Rules are sorted on their righthand sides.)

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

[−,+,+,#,A] → [*,A,B,#,+]	(66)
[+,+,+,#,A] → [*,A,C,#,+]	(66)
[A,+,+,−,+] → [*,A,C,#,B]	(44)
[B,+,#,#,−] → [*,B,#,#,#]	(27)
[B,+,#,#,*] → [*,B,#,#,−]	(20)
[B,+,#,−,+] → [*,B,#,#,A]	(45)
[−,+,#,A,+] → [*,B,#,+,A]	(66)
[B,+,#,*,#] → [*,B,#,−,#]	(19)
[B,+,#,*,+] → [*,B,#,−,B]	(39)
[B,+,*,#,#] → [*,B,−,#,#]	(21)
[B,+,*,#,+] → [*,B,−,#,B]	(40)
[B,+,*,+,+] → [*,B,−,B,B]	(38)
[−,+,+,A,#] → [*,B,A,+,#]	(67)
[A,+,+,+,−] → [*,B,A,C,#]	(44)
[B,+,+,#,*] → [*,B,B,#,−]	(39)
[B,+,+,*,#] → [*,B,B,−,#]	(40)
[B,+,+,*,+] → [*,B,B,−,B]	(38)
[B,+,+,+,*] → [*,B,B,B,−]	(38)
[C,+,−,#,#] → [*,C,#,#,#]	(28)
[C,+,#,#,*] → [*,C,#,#,−]	(20)
[+,+,#,A,+] → [*,C,#,+,A]	(66)
[C,+,#,*,#] → [*,C,#,−,#]	(19)
[C,+,#,*,+] → [*,C,#,−,C]	(39)
[A,+,−,+,+] → [*,C,#,B,A]	(44)
[−,+,B,#,A] → [*,C,*,#,+]	(64)
[−,+,A,#,B] → [*,C,+,#,*]	(65)
[C,+,*,#,#] → [*,C,−,#,#]	(21)
[C,+,*,#,+] → [*,C,−,#,C]	(40)
[C,+,*,+,+] → [*,C,−,C,C]	(38)
[C,+,+,−,#] → [*,C,A,#,#]	(46)
[+,+,+,A,#] → [*,C,A,+,#]	(67)
[C,+,+,#,*] → [*,C,C,#,−]	(39)
[C,+,+,*,#] → [*,C,C,−,#]	(40)
[C,+,+,*,+] → [*,C,C,−,C]	(38)
[C,+,+,+,*] → [*,C,C,C,−]	(38)
[*,#,#,#,+] → [+,#,#,#,*]	(29)
[#,#,#,#,−] → [+,#,#,#,+]	(13)
[*,#,#,+,#] → [+,#,#,*,#]	(29)
[*,#,#,+,+] → [+,#,#,*,*]	(48)
[#,#,#,−,#] → [+,#,#,+,#]	(13)
[D,#,#,C,+] → [+,#,#,+,A]	(69)
[D,#,#,+,C] → [+,#,#,A,+]	(70)
[*,#,+,#,#] → [+,#,*,#,#]	(29)
[*,#,+,+,#] → [+,#,*,*,#]	(48)
[*,#,+,+,+] → [+,#,*,*,*]	(47)
[#,#,−,#,#] → [+,#,+,#,#]	(13)
[D,#,C,#,+] → [+,#,+,#,A]	(68)
[*,#,B,#,#] → [+,#,+,#,B]	(30)
[D,#,C,+,#] → [+,#,+,A,#]	(69)
[*,#,C,+,+] → [+,#,+,A,A]	(66)
[−,#,C,+,+] → [+,#,+,A,B]	(66)
[+,#,C,+,+] → [+,#,+,A,C]	(66)
[D,#,+,#,C] → [+,#,A,#,+]	(68)
[D,#,+,C,#] → [+,#,A,+,#]	(70)
[#,#,+,*,#] → [+,#,A,+,D]	(88)
[*,#,+,+,C] → [+,#,A,A,+]	(67)
[*,#,#,#,B] → [+,#,B,#,+]	(30)
[−,#,+,+,C] → [+,#,B,A,+]	(67)
[+,#,+,+,C] → [+,#,C,A,+]	(67)
[#,#,#,*,+] → [+,#,D,+,A]	(87)
[*,+,#,#,#] → [+,*,#,#,#]	(29)
[*,+,#,#,+] → [+,*,#,#,*]	(48)
[*,+,#,+,+] → [+,*,#,*,*]	(47)
[*,+,+,#,#] → [+,*,*,#,#]	(48)
[*,+,+,#,+] → [+,*,*,#,*]	(47)
[*,+,+,+,#] → [+,*,*,*,#]	(47)
[#,−,#,#,#] → [+,+,#,#,#]	(13)
[D,C,#,#,+] → [+,+,#,#,A]	(70)
[D,C,#,+,#] → [+,+,#,A,#]	(68)
[*,C,#,+,+] → [+,+,#,A,A]	(67)
[*,B,#,#,#] → [+,+,#,B,#]	(30)
[−,C,#,+,+] → [+,+,#,B,A]	(67)
[+,C,#,+,+] → [+,+,#,C,A]	(67)
[D,C,+,#,#] → [+,+,A,#,#]	(69)
[#,*,+,#,#] → [+,+,A,#,D]	(87)
[*,C,+,+,#] → [+,+,A,A,#]	(66)
[−,C,+,+,#] → [+,+,A,B,#]	(66)
[+,C,+,+,#] → [+,+,A,C,#]	(66)
[#,*,#,#,+] → [+,+,D,#,A]	(88)
[D,+,#,#,C] → [+,A,#,#,+]	(69)
[D,+,#,C,#] → [+,A,#,+,#]	(68)
[*,+,#,C,+] → [+,A,#,+,A]	(66)
[#,+,#,#,*] → [+,A,#,D,+]	(87)
[D,+,C,#,#] → [+,A,+,#,#]	(70)
[*,+,C,#,+] → [+,A,+,#,A]	(67)
[−,+,C,#,+] → [+,A,+,#,B]	(67)
[+,+,C,#,+] → [+,A,+,#,C]	(67)
[#,+,*,#,#] → [+,A,+,D,#]	(88)
[*,+,+,#,C] → [+,A,A,#,+]	(66)
[*,+,+,C,#] → [+,A,A,+,#]	(67)
[−,+,+,#,C] → [+,A,B,#,+]	(66)
[+,+,+,#,C] → [+,A,C,#,+]	(66)
[*,#,#,B,#] → [+,B,#,+,#]	(30)
[−,+,#,C,+] → [+,B,#,+,A]	(66)
[−,+,+,C,#] → [+,B,A,+,#]	(67)
[+,+,#,C,+] → [+,C,#,+,A]	(66)
[+,+,+,C,#] → [+,C,A,+,#]	(67)
[#,#,#,+,*] → [+,D,#,A,+]	(88)
[#,#,*,+,#] → [+,D,+,A,#]	(87)
[+,#,#,#,+] → [−,#,#,#,*]	(18)
[*,#,#,#,A] → [−,#,#,#,+]	(9)
[+,#,#,+,#] → [−,#,#,*,#]	(18)
[+,#,#,+,+] → [−,#,#,*,*]	(50)
[*,#,#,A,#] → [−,#,#,+,#]	(9)
[+,#,#,A,+] → [−,#,#,+,B]	(72)
[+,#,#,+,A] → [−,#,#,C,+]	(73)
[+,#,+,#,#] → [−,#,*,#,#]	(18)
[+,#,+,+,#] → [−,#,*,*,#]	(50)
[+,#,+,+,+] → [−,#,*,*,*]	(49)
[*,#,A,#,#] → [−,#,+,#,#]	(9)
[+,#,A,#,+] → [−,#,+,#,A]	(71)
[*,#,B,+,+] → [−,#,+,A,A]	(66)
[−,#,B,+,+] → [−,#,+,A,B]	(66)
[+,#,B,+,+] → [−,#,+,A,C]	(66)
[+,#,A,+,#] → [−,#,+,B,#]	(72)
[#,#,A,+,#] → [−,#,+,B,B]	(60)
[−,#,A,+,D] → [−,#,+,D,−]	(62)
[−,#,D,+,A] → [−,#,−,D,+]	(63)
[+,#,+,#,A] → [−,#,A,#,+]	(71)
[*,#,+,+,B] → [−,#,A,A,+]	(67)
[−,#,+,+,B] → [−,#,B,A,+]	(67)
[#,#,#,+,A] → [−,#,B,C,+]	(61)
[+,#,+,A,#] → [−,#,C,+,#]	(73)
[+,#,+,+,B] → [−,#,C,A,+]	(67)
[+,+,#,#,#] → [−,*,#,#,#]	(18)
[+,+,#,#,+] → [−,*,#,#,*]	(50)
[+,+,#,+,+] → [−,*,#,*,*]	(49)
[+,+,+,#,#] → [−,*,*,#,#]	(50)
[+,+,+,#,+] → [−,*,*,#,*]	(49)
[+,+,+,+,#] → [−,*,*,*,#]	(49)
[*,A,#,#,#] → [−,+,#,#,#]	(9)
[+,A,#,#,+] → [−,+,#,#,C]	(73)
[−,A,#,D,+] → [−,+,#,−,D]	(63)
[+,A,#,+,#] → [−,+,#,A,#]	(71)
[*,B,#,+,+] → [−,+,#,A,A]	(67)
[−,B,#,+,+] → [−,+,#,B,A]	(67)
[#,A,#,#,+] → [−,+,#,B,C]	(61)
[+,B,#,+,+] → [−,+,#,C,A]	(67)
[*,B,+,+,#] → [−,+,A,A,#]	(66)
[−,B,+,+,#] → [−,+,A,B,#]	(66)
[+,B,+,+,#] → [−,+,A,C,#]	(66)
[+,A,+,#,#] → [−,+,B,#,#]	(72)
[#,A,+,#,#] → [−,+,B,B,#]	(60)
[−,A,+,D,#] → [−,+,D,−,#]	(62)
[−,D,#,A,+] → [−,−,#,+,D]	(62)
[−,D,+,A,#] → [−,−,D,+,#]	(63)
[+,+,#,A,#] → [−,A,#,+,#]	(71)
[*,+,#,B,+] → [−,A,#,+,A]	(66)
[*,+,B,#,+] → [−,A,+,#,A]	(67)
[−,+,B,#,+] → [−,A,+,#,B]	(67)
[+,+,B,#,+] → [−,A,+,#,C]	(67)
[*,+,+,#,B] → [−,A,A,#,+]	(66)
[*,+,+,B,#] → [−,A,A,+,#]	(67)
[−,+,+,#,B] → [−,A,B,#,+]	(66)
[+,+,+,#,B] → [−,A,C,#,+]	(66)
[+,+,#,#,A] → [−,B,#,#,+]	(72)
[−,+,#,B,+] → [−,B,#,+,A]	(66)
[#,#,#,A,+] → [−,B,#,+,B]	(60)
[−,+,+,B,#] → [−,B,A,+,#]	(67)
[#,+,#,#,A] → [−,B,B,#,+]	(60)
[#,#,+,A,#] → [−,B,C,+,#]	(61)
[+,+,#,B,+] → [−,C,#,+,A]	(66)
[+,+,A,#,#] → [−,C,+,#,#]	(73)
[#,+,A,#,#] → [−,C,+,#,B]	(61)
[+,+,+,B,#] → [−,C,A,+,#]	(67)
[−,+,A,#,D] → [−,D,+,#,−]	(63)
[−,+,D,#,A] → [−,D,−,#,+]	(62)
[*,#,*,#,+] → [A,#,#,#,A]	(23)
[C,#,#,#,A] → [A,#,#,*,+]	(16)
[B,#,#,A,#] → [A,#,#,+,*]	(15)
[A,#,#,A,+] → [A,#,#,+,A]	(3)
[B,#,#,A,+] → [A,#,#,+,B]	(3)
[C,#,#,A,+] → [A,#,#,+,C]	(3)
[*,*,#,+,#] → [A,#,#,A,#]	(23)
[A,#,#,+,A] → [A,#,#,A,+]	(4)
[B,#,#,+,A] → [A,#,#,B,+]	(4)
[C,#,#,+,A] → [A,#,#,C,+]	(4)
[A,#,#,#,A] → [A,#,*,#,+]	(14)
[C,#,B,#,A] → [A,#,*,*,+]	(37)
[C,#,#,A,#] → [A,#,*,+,#]	(16)
[A,#,A,#,#] → [A,#,+,#,*]	(14)
[−,#,A,#,#] → [A,#,+,#,−]	(10)
[A,#,A,#,+] → [A,#,+,#,A]	(2)
[B,#,A,#,+] → [A,#,+,#,B]	(2)
[C,#,A,#,+] → [A,#,+,#,C]	(2)
[B,#,A,#,#] → [A,#,+,*,#]	(15)
[B,#,A,#,B] → [A,#,+,*,*]	(36)
[A,#,A,+,#] → [A,#,+,A,#]	(3)
[#,#,A,+,−] → [A,#,+,A,−]	(83)
[A,#,A,+,+] → [A,#,+,A,A]	(5)
[B,#,A,+,#] → [A,#,+,B,#]	(3)
[B,#,A,+,+] → [A,#,+,B,B]	(5)
[C,#,A,+,#] → [A,#,+,C,#]	(3)
[C,#,A,+,+] → [A,#,+,C,C]	(5)
[−,#,#,#,A] → [A,#,−,#,+]	(10)
[#,#,−,+,A] → [A,#,−,A,+]	(84)
[*,#,+,#,*] → [A,#,A,#,#]	(23)
[A,#,+,#,A] → [A,#,A,#,+]	(2)
[A,#,+,A,#] → [A,#,A,+,#]	(4)
[A,#,+,A,+] → [A,#,A,+,A]	(7)
[A,#,+,+,A] → [A,#,A,A,+]	(6)
[B,#,+,#,A] → [A,#,B,#,+]	(2)
[B,#,+,A,#] → [A,#,B,+,#]	(4)
[B,#,+,A,+] → [A,#,B,+,B]	(7)
[B,#,+,+,A] → [A,#,B,B,+]	(6)
[*,*,+,+,+] → [A,#,B,B,B]	(41)
[C,#,+,#,A] → [A,#,C,#,+]	(2)
[C,#,+,A,#] → [A,#,C,+,#]	(4)
[C,#,+,A,+] → [A,#,C,+,C]	(7)
[C,#,+,+,A] → [A,#,C,C,+]	(6)
[B,#,#,#,A] → [A,*,#,#,+]	(15)
[A,#,#,A,#] → [A,*,#,+,#]	(14)
[B,B,#,A,#] → [A,*,#,+,*]	(36)
[B,#,B,#,A] → [A,*,*,#,+]	(36)
[A,#,B,#,A] → [A,*,*,*,+]	(35)
[C,B,#,A,#] → [A,*,*,+,#]	(37)
[A,B,#,A,#] → [A,*,*,+,*]	(35)
[C,#,A,#,#] → [A,*,+,#,#]	(16)
[C,#,A,#,B] → [A,*,+,#,*]	(37)
[A,#,A,#,B] → [A,*,+,*,*]	(35)
[C,A,#,#,#] → [A,+,#,#,*]	(16)
[A,A,#,#,+] → [A,+,#,#,A]	(4)
[B,A,#,#,+] → [A,+,#,#,B]	(4)
[C,A,#,#,+] → [A,+,#,#,C]	(4)
[A,A,#,#,#] → [A,+,#,*,#]	(14)
[C,A,#,B,#] → [A,+,#,*,*]	(37)
[−,A,#,#,#] → [A,+,#,−,#]	(10)
[#,A,#,−,+] → [A,+,#,−,A]	(84)

[A,A,#,+,#] → [A,+,#,A,#] (2)
[A,A,#,+,+] → [A,+,#,A,A] (6)
[B,A,#,+,#] → [A,+,#,B,#] (2)
[B,A,#,+,+] → [A,+,#,B,B] (6)
[C,A,#,+,#] → [A,+,#,C,#] (2)
[C,A,#,+,+] → [A,+,#,C,C] (6)
[B,A,#,#,#] → [A,+,*,#,#] (15)
[B,A,#,B,#] → [A,+,*,*,#] (36)
[A,A,#,B,#] → [A,+,*,*,*] (35)
[+,A,#,#,#] → [A,+,−,−,−] (32)
[A,A,+,#,#] → [A,+,A,#,#] (3)
[A,A,+,#,+] → [A,+,A,#,A] (7)
[#,A,+,−,#] → [A,+,A,−,#] (83)
[A,A,+,+,#] → [A,+,A,A,#] (5)
[A,A,+,+,+] → [A,+,A,A,A] (8)
[B,A,+,#,#] → [A,+,B,#,#] (3)
[B,A,+,#,+] → [A,+,B,#,B] (7)
[B,A,+,+,#] → [A,+,B,B,#] (5)
[B,A,+,+,+] → [A,+,B,B,B] (8)
[C,A,+,#,#] → [A,+,C,#,#] (3)
[C,A,+,#,+] → [A,+,C,#,C] (7)
[C,A,+,+,#] → [A,+,C,C,#] (5)
[C,A,+,+,+] → [A,+,C,C,C] (8)
[−,#,#,A,#] → [A,−,#,+,#] (10)
[#,−,#,A,+] → [A,−,#,+,A] (83)
[+,#,A,#,#] → [A,−,+,−,−] (32)
[+,#,#,A,#] → [A,−,−,+,−] (32)
[+,#,#,#,A] → [A,−,−,−,+] (32)
[#,−,+,A,#] → [A,−,A,+,#] (84)
[*,+,#,*,#] → [A,A,#,#,#] (23)
[A,+,#,#,A] → [A,A,#,#,+] (3)
[A,+,#,A,#] → [A,A,#,+,#] (2)
[A,+,#,A,+] → [A,A,#,+,A] (5)
[A,+,#,+,A] → [A,A,#,A,+] (7)
[A,+,A,#,#] → [A,A,+,#,#] (4)
[#,+,A,#,−] → [A,A,+,#,−] (84)
[A,+,A,#,+] → [A,A,+,#,A] (6)
[A,+,A,+,#] → [A,A,+,A,#] (7)
[A,+,A,+,+] → [A,A,+,A,A] (8)
[#,+,−,#,A] → [A,A,−,#,+] (83)
[A,+,+,#,A] → [A,A,A,#,+] (5)
[A,+,+,A,#] → [A,A,A,+,#] (6)
[A,+,+,A,+] → [A,A,A,+,A] (8)
[A,+,+,+,A] → [A,A,A,A,+] (8)
[B,+,#,#,A] → [A,B,#,#,+] (3)
[B,+,#,A,#] → [A,B,#,+,#] (2)
[B,+,#,A,+] → [A,B,#,+,B] (5)
[B,+,#,+,A] → [A,B,#,B,+] (7)
[*,+,*,+,+] → [A,B,#,B,B] (41)
[B,+,A,#,#] → [A,B,+,#,#] (4)
[B,+,A,#,+] → [A,B,+,#,B] (6)
[B,+,A,+,#] → [A,B,+,B,#] (7)
[B,+,A,+,+] → [A,B,+,B,B] (8)
[B,+,+,#,A] → [A,B,B,#,+] (5)
[*,+,+,*,+] → [A,B,B,#,B] (41)
[B,+,+,A,#] → [A,B,B,+,#] (6)
[B,+,+,A,+] → [A,B,B,+,B] (8)
[*,+,+,+,*] → [A,B,B,B,#] (41)
[B,+,+,+,A] → [A,B,B,B,+] (8)
[C,+,#,#,A] → [A,C,#,#,+] (3)
[C,+,#,A,#] → [A,C,#,+,#] (2)
[C,+,#,A,+] → [A,C,#,+,C] (5)
[C,+,#,+,A] → [A,C,#,C,+] (7)
[C,+,A,#,#] → [A,C,+,#,#] (4)
[C,+,A,#,+] → [A,C,+,#,C] (6)
[C,+,A,+,#] → [A,C,+,C,#] (7)
[C,+,A,+,+] → [A,C,+,C,C] (8)
[C,+,+,#,A] → [A,C,C,#,+] (5)
[C,+,+,A,#] → [A,C,C,+,#] (6)
[C,+,+,A,+] → [A,C,C,+,C] (8)
[C,+,+,+,A] → [A,C,C,C,+] (8)
[*,#,#,*,+] → [B,#,#,#,A] (24)
[C,#,#,#,B] → [B,#,#,*,+] (16)
[B,#,#,B,#] → [B,#,#,+,*] (15)
[−,#,#,B,#] → [B,#,#,+,−] (11)
[A,#,#,B,+] → [B,#,#,+,A] (3)

[B,#,#,B,+] → [B,#,#,+,B] (3)
[C,#,#,B,+] → [B,#,#,+,C] (3)
[#,#,D,B,+] → [B,#,#,+,D] (91)
[*,#,*,+,#] → [B,#,#,A,#] (24)
[A,#,#,+,B] → [B,#,#,A,+] (4)
[*,#,*,+,+] → [B,#,#,A,B] (42)
[B,#,#,+,B] → [B,#,#,B,+] (4)
[C,#,#,+,B] → [B,#,#,C,+] (4)
[#,D,#,+,C] → [B,#,#,D,+] (92)
[A,#,#,#,B] → [B,#,*,#,+] (14)
[C,#,#,B,#] → [B,#,*,+,#] (16)
[A,#,B,#,#] → [B,#,+,#,*] (14)
[A,#,B,#,+] → [B,#,+,#,A] (2)
[B,#,B,#,+] → [B,#,+,#,B] (2)
[C,#,B,#,+] → [B,#,+,#,C] (2)
[D,#,B,#,+] → [B,#,+,#,D] (56)
[B,#,B,#,#] → [B,#,+,*,#] (15)
[−,#,B,#,#] → [B,#,+,−,#] (11)
[+,#,B,#,#] → [B,#,+,−,−] (33)
[A,#,B,+,#] → [B,#,+,A,#] (3)
[A,#,B,+,+] → [B,#,+,A,A] (5)
[B,#,B,+,#] → [B,#,+,B,#] (3)
[B,#,B,+,+] → [B,#,+,B,B] (5)
[C,#,B,+,#] → [B,#,+,C,#] (3)
[C,#,B,+,+] → [B,#,+,C,C] (5)
[#,D,B,+,#] → [B,#,+,D,#] (91)
[*,*,+,#,#] → [B,#,A,#,#] (24)
[A,#,+,#,B] → [B,#,A,#,+] (2)
[A,#,+,B,#] → [B,#,A,+,#] (4)
[A,#,+,B,+] → [B,#,A,+,A] (7)
[A,#,+,+,B] → [B,#,A,A,+] (6)
[*,*,+,+,#] → [B,#,A,B,#] (42)
[B,#,+,#,B] → [B,#,B,#,+] (2)
[B,#,+,B,#] → [B,#,B,+,#] (4)
[B,#,+,B,+] → [B,#,B,+,B] (7)
[B,#,+,+,B] → [B,#,B,B,+] (6)
[C,#,+,#,B] → [B,#,C,#,+] (2)
[C,#,+,B,#] → [B,#,C,+,#] (4)
[C,#,+,B,+] → [B,#,C,+,C] (7)
[C,#,+,+,B] → [B,#,C,C,+] (6)
[D,#,+,#,B] → [B,#,D,#,+] (56)
[#,#,+,C,D] → [B,#,D,+,#] (92)
[B,#,#,#,B] → [B,*,#,#,+] (15)
[A,#,#,B,#] → [B,*,#,+,#] (14)
[C,#,B,#,#] → [B,*,+,#,#] (16)
[C,B,#,#,#] → [B,+,#,#,*] (16)
[A,B,#,#,+] → [B,+,#,#,A] (4)
[B,B,#,#,+] → [B,+,#,#,B] (4)
[C,B,#,#,+] → [B,+,#,#,C] (4)
[#,C,D,#,+] → [B,+,#,#,D] (92)
[A,B,#,#,#] → [B,+,#,*,#] (14)
[A,B,#,+,#] → [B,+,#,A,#] (2)
[A,B,#,+,+] → [B,+,#,A,A] (6)
[B,B,#,+,#] → [B,+,#,B,#] (2)
[B,B,#,+,+] → [B,+,#,B,B] (6)
[C,B,#,+,#] → [B,+,#,C,#] (2)
[C,B,#,+,+] → [B,+,#,C,C] (6)
[D,B,#,+,#] → [B,+,#,D,#] (56)
[B,B,#,#,#] → [B,+,*,#,#] (15)
[−,B,#,#,#] → [B,+,−,#,#] (11)
[+,B,#,#,#] → [B,+,−,−,#] (33)
[A,B,+,#,#] → [B,+,A,#,#] (3)
[A,B,+,#,+] → [B,+,A,#,A] (7)
[A,B,+,+,#] → [B,+,A,A,#] (5)
[A,B,+,+,+] → [B,+,A,A,A] (8)
[B,B,+,#,#] → [B,+,B,#,#] (3)
[B,B,+,#,+] → [B,+,B,#,B] (7)
[B,B,+,+,#] → [B,+,B,B,#] (5)
[B,B,+,+,+] → [B,+,B,B,B] (8)
[C,B,+,#,#] → [B,+,C,#,#] (3)
[C,B,+,#,+] → [B,+,C,#,C] (7)
[C,B,+,+,#] → [B,+,C,C,#] (5)
[C,B,+,+,+] → [B,+,C,C,C] (8)
[#,B,+,#,D] → [B,+,D,#,#] (91)
[−,#,#,#,B] → [B,−,#,#,+] (11)
[+,#,#,B,#] → [B,−,#,+,−] (33)

[+,#,#,#,B] → [B,−,−,#,+] (33)
[*,+,#,#,*] → [B,A,#,#,#] (24)
[A,+,#,#,B] → [B,A,#,#,+] (3)
[A,+,#,B,#] → [B,A,#,+,#] (2)
[A,+,#,B,+] → [B,A,#,+,A] (5)
[A,+,#,+,B] → [B,A,#,A,+] (7)
[A,+,B,#,#] → [B,A,+,#,#] (4)
[A,+,B,#,+] → [B,A,+,#,A] (6)
[A,+,B,+,#] → [B,A,+,A,#] (7)
[A,+,B,+,+] → [B,A,+,A,A] (8)
[A,+,+,#,B] → [B,A,A,#,+] (5)
[A,+,+,B,#] → [B,A,A,+,#] (6)
[A,+,+,B,+] → [B,A,A,+,A] (8)
[A,+,+,+,B] → [B,A,A,A,+] (8)
[*,+,+,#,*] → [B,A,B,#,#] (42)
[B,+,#,#,B] → [B,B,#,#,+] (3)
[*,+,#,*,+] → [B,B,#,#,A] (42)
[B,+,#,B,#] → [B,B,#,+,#] (2)
[B,+,#,B,+] → [B,B,#,+,B] (5)
[B,+,#,+,B] → [B,B,#,B,+] (7)
[B,+,B,#,#] → [B,B,+,#,#] (4)
[B,+,B,#,+] → [B,B,+,#,B] (6)
[B,+,B,+,#] → [B,B,+,B,#] (7)
[B,+,B,+,+] → [B,B,+,B,B] (8)
[B,+,+,#,B] → [B,B,B,#,+] (5)
[B,+,+,B,#] → [B,B,B,+,#] (6)
[B,+,+,B,+] → [B,B,B,+,B] (8)
[B,+,+,+,B] → [B,B,B,B,+] (8)
[C,+,#,#,B] → [B,C,#,#,+] (3)
[C,+,#,B,#] → [B,C,#,+,#] (2)
[C,+,#,B,+] → [B,C,#,+,C] (5)
[C,+,#,+,B] → [B,C,#,C,+] (7)
[C,+,B,#,#] → [B,C,+,#,#] (4)
[C,+,B,#,+] → [B,C,+,#,C] (6)
[C,+,B,+,#] → [B,C,+,C,#] (7)
[C,+,B,+,+] → [B,C,+,C,C] (8)
[C,+,+,#,B] → [B,C,C,#,+] (5)
[C,+,+,B,#] → [B,C,C,+,#] (6)
[C,+,+,B,+] → [B,C,C,+,C] (8)
[C,+,+,+,B] → [B,C,C,C,+] (8)
[#,+,#,D,B] → [B,D,#,#,+] (91)
[D,+,#,B,#] → [B,D,#,+,#] (56)
[#,+,C,D,#] → [B,D,+,#,#] (92)
[*,*,#,#,+] → [C,#,#,#,A] (25)
[C,#,#,#,C] → [C,#,#,*,+] (16)
[B,#,#,C,#] → [C,#,#,+,*] (15)
[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)

Appendix C

Transition rules for SR_9

[#,#,#,#,#,#,#] → [#,#,#,#,#,#,#]	(1)	[B,#,+,=,B,+,+] → [B,#,B,+,+,B,B]	(10)
[A,#,+,=,A,#,#] → [A,#,A,=,+,#,#]	(2)	[B,#,+,=,C,+,+] → [C,#,B,+,+,B,B]	(10)
[A,#,+,=,B,#,#] → [B,#,A,=,+,#,#]	(2)	[C,#,+,=,A,+,+] → [A,#,C,+,+,C,C]	(10)
[A,#,+,=,C,#,#] → [C,#,A,=,+,#,#]	(2)	[C,#,+,=,B,+,+] → [B,#,C,+,+,C,C]	(10)
[B,#,+,=,A,#,#] → [A,#,B,=,+,#,#]	(2)	[C,#,+,=,C,+,+] → [C,#,C,+,+,C,C]	(10)
[B,#,+,=,B,#,#] → [B,#,B,=,+,#,#]	(2)	[A,#,+,=,A,#,+] → [A,#,A,=,+,#,A]	(7)
[B,#,+,=,C,#,#] → [C,#,B,=,+,#,#]	(2)	[A,#,+,=,B,#,+] → [B,#,A,=,+,#,A]	(7)
[C,#,+,=,A,#,#] → [A,#,C,=,+,#,#]	(2)	[A,#,+,=,C,#,+] → [C,#,A,=,+,#,A]	(7)
[C,#,+,=,B,#,#] → [B,#,C,=,+,#,#]	(2)	[B,#,+,=,A,#,+] → [A,#,B,=,+,#,B]	(7)
[C,#,+,=,C,#,#] → [C,#,C,=,+,#,#]	(2)	[B,#,+,=,B,#,+] → [B,#,B,=,+,#,B]	(7)
[A,#,#,=,A,#,+] → [A,#,#,=,+,#,A]	(3)	[B,#,+,=,C,#,+] → [C,#,B,=,+,#,B]	(7)
[A,#,#,=,B,#,+] → [B,#,#,=,+,#,A]	(3)	[C,#,+,=,A,#,+] → [A,#,C,=,+,#,C]	(7)
[A,#,#,=,C,#,+] → [C,#,#,=,+,#,A]	(3)	[C,#,+,=,B,#,+] → [B,#,C,=,+,#,C]	(7)
[B,#,#,=,A,#,+] → [A,#,#,=,+,#,B]	(3)	[C,#,+,=,C,#,+] → [C,#,C,=,+,#,C]	(7)
[B,#,#,=,B,#,+] → [B,#,#,=,+,#,B]	(3)	[A,#,+,=,A,+,#] → [A,#,A,=,+,A,#]	(8)
[B,#,#,=,C,#,+] → [C,#,#,=,+,#,B]	(3)	[A,#,+,=,B,+,#] → [B,#,A,=,+,A,#]	(8)
[C,#,#,=,A,#,+] → [A,#,#,=,+,#,C]	(3)	[A,#,+,=,C,+,#] → [C,#,A,=,+,A,#]	(8)
[C,#,#,=,B,#,+] → [B,#,#,=,+,#,C]	(3)	[B,#,+,=,A,+,#] → [A,#,B,=,+,B,#]	(8)
[C,#,#,=,C,#,+] → [C,#,#,=,+,#,C]	(3)	[B,#,+,=,B,+,#] → [B,#,B,=,+,B,#]	(8)
[A,#,#,=,A,+,#] → [A,#,#,=,+,A,#]	(4)	[B,#,+,=,C,+,#] → [C,#,B,=,+,B,#]	(8)
[A,#,#,=,B,+,#] → [B,#,#,=,+,A,#]	(4)	[C,#,+,=,A,+,#] → [A,#,C,=,+,C,#]	(8)
[A,#,#,=,C,+,#] → [C,#,#,=,+,A,#]	(4)	[C,#,+,=,B,+,#] → [B,#,C,=,+,C,#]	(8)
[B,#,#,=,A,+,#] → [A,#,#,=,+,B,#]	(4)	[C,#,+,=,C,+,#] → [C,#,C,=,+,C,#]	(8)
[B,#,#,=,B,+,#] → [B,#,#,=,+,B,#]	(4)	[#,=,+,=,=,#,#] → [#,=,=,=,+,#,#]	(11)
[B,#,#,=,C,+,#] → [C,#,#,=,+,B,#]	(4)	[#,=,#,=,=,#,+] → [#,=,#,=,+,#,=]	(12)
[C,#,#,=,A,+,#] → [A,#,#,=,+,C,#]	(4)	[#,=,+,=,=,#,+] → [#,=,=,=,+,#,=]	(13)
[C,#,#,=,B,+,#] → [B,#,#,=,+,C,#]	(4)	[#,+,+,+,=,+,+] → [#,=,=,=,+,=,=]	(14)
[C,#,#,=,C,+,#] → [C,#,#,=,+,C,#]	(4)	[#,=,#,#,#,#,#] → [#,*,#,#,#,#,#]	(15)
[A,#,+,#,A,#,#] → [A,#,A,#,+,#,#]	(5)	[#,=,=,#,#,#,#] → [#,*,*,#,#,#,#]	(16)
[A,#,+,#,B,#,#] → [B,#,A,#,+,#,#]	(5)	[*,*,#,=,A,*,*] → [−,#,#,=,+,#,#]	(17)
[A,#,+,#,C,#,#] → [C,#,A,#,+,#,#]	(5)	[*,=,#,=,A,#,#] → [A,=,#,=,−,#,#]	(18)
[B,#,+,#,A,#,#] → [A,#,B,#,+,#,#]	(5)	[#,=,−,=,=,#,#] → [#,=,−,=,+,#,#]	(19)
[B,#,+,#,B,#,#] → [B,#,B,#,+,#,#]	(5)	[#,=,#,=,=,#,−] → [#,=,#,=,+,#,−]	(20)
[B,#,+,#,C,#,#] → [C,#,B,#,+,#,#]	(5)	[#,=,−,=,=,#,−] → [#,=,−,=,+,#,−]	(21)
[C,#,+,#,A,#,#] → [A,#,C,#,+,#,#]	(5)	[#,+,−,+,=,−,−] → [#,=,−,=,+,−,−]	(22)
[C,#,+,#,B,#,#] → [B,#,C,#,+,#,#]	(5)	[−,*,#,=,A,*,*] → [A,#,−,A,+,#,#]	(23)
[C,#,+,#,C,#,#] → [C,#,C,#,+,#,#]	(5)	[−,*,#,=,B,*,*] → [B,#,#,B,+,#,−]	(24)
[A,#,#,#,A,#,+] → [A,#,#,#,+,#,A]	(6)	[−,*,#,=,C,*,*] → [C,#,#,C,+,−,#]	(25)
[A,#,#,#,B,#,+] → [B,#,#,#,+,#,A]	(6)	[A,=,#,=,−,#,#] → [A,=,#,=,+,#,#]	(26)
[A,#,#,#,C,#,+] → [C,#,#,#,+,#,A]	(6)	[#,#,#,#,−,#,#] → [+,=,#,=,+,=,=]	(27)
[B,#,#,#,A,#,+] → [A,#,#,#,+,#,B]	(6)	[A,#,#,=,A,#,#] → [A,#,*,A,+,#,#]	(28)
[B,#,#,#,B,#,+] → [B,#,#,#,+,#,B]	(6)	[B,#,#,=,A,#,#] → [A,#,#,B,+,#,*]	(29)
[B,#,#,#,C,#,+] → [C,#,#,#,+,#,B]	(6)	[C,#,#,=,A,#,#] → [A,#,#,C,+,*,#]	(30)
[C,#,#,#,A,#,+] → [A,#,#,#,+,#,C]	(6)	[A,#,#,=,B,#,#] → [B,#,*,−,+,#,#]	(31)
[C,#,#,#,B,#,+] → [B,#,#,#,+,#,C]	(6)	[B,#,#,=,B,#,#] → [B,#,#,−,+,#,*]	(32)
[C,#,#,#,C,#,+] → [C,#,#,#,+,#,C]	(6)	[C,#,#,=,B,#,#] → [B,#,#,−,+,*,#]	(33)
[A,#,+,=,A,+,+] → [A,#,A,+,+,A,A]	(10)	[A,A,#,A,−,#,#] → [+,=,A,=,+,#,#]	(34)
[A,#,+,=,B,+,+] → [B,#,A,+,+,A,A]	(10)	[A,B,#,C,−,#,#] → [+,=,#,=,+,#,A]	(35)
[A,#,+,=,C,+,+] → [C,#,A,+,+,A,A]	(10)	[A,B,#,B,−,#,#] → [B,=,#,=,+,−,−]	(41)
[B,#,+,=,A,+,+] → [A,#,B,+,+,B,B]	(10)	[A,C,#,C,−,#,#] → [C,=,#,=,+,−,−]	(42)

[+,#,#,#,*,#,#]	→	[*,=,#,=,+,=,=]	(36)
[+,A,#,A,=,#,#]	→	[#,=,A,=,+,#,#]	(37)
[+,B,#,C,=,#,#]	→	[#,=,#,=,+,#,A]	(38)
[B,B,#,B,=,#,#]	→	[B,=,#,=,+,*,*]	(43)
[C,C,#,C,=,#,#]	→	[C,=,#,=,+,*,*]	(43)
[#,=,#,=,A,#,#]	→	[*,=,#,=,−,#,#]	(39)
[#,=,#,#,#,−,#]	→	[=,=,#,#,#,+,#]	(44)
[+,−,#,−,=,#,#]	→	[#,=,B,=,+,#,#]	(40)
[=,=,#,#,#,*,#]	→	[=,−,#,#,#,=,#]	(45)
[B,=,#,=,=,+,+]	→	[B,=,#,=,+,+,+]	(46)
[C,=,#,=,=,+,+]	→	[C,=,#,=,+,+,+]	(46)
[−,*,*,−,*,A,#]	→	[A,#,#,−,#,+,#]	(47)
[=,=,#,#,#,+,#]	→	[A,+,#,#,#,=,#]	(48)
[B,=,#,=,=,=,=]	→	[B,=,#,=,+,=,=]	(49)
[C,=,#,=,=,=,=]	→	[C,=,#,=,+,=,=]	(49)
[A,−,#,#,#,=,#]	→	[+,+,#,#,#,=,#]	(50)
[+,A,#,#,#,=,#]	→	[*,+,#,#,#,B,#]	(51)
[*,A,#,#,#,=,#]	→	[*,+,#,#,#,C,#]	(52)
[B,=,#,=,=,B,B]	→	[B,=,#,=,+,A,A]	(53)
[C,=,#,=,=,B,B]	→	[C,=,#,=,+,A,A]	(53)
[*,C,#,#,#,A,#]	→	[C,+,−,#,#,=,#]	(54)
[*,B,#,#,#,A,#]	→	[B,+,#,#,−,=,#]	(55)
[B,=,#,=,=,C,C]	→	[=,=,#,=,+,B,B]	(56)
[C,=,#,=,=,C,C]	→	[=,=,#,=,+,C,C]	(56)
[C,A,#,#,#,B,#]	→	[A,+,*,#,#,=,#]	(57)
[B,A,#,#,#,C,#]	→	[A,+,#,#,*,=,#]	(58)
[=,=,#,=,=,=,=]	→	[A,=,A,=,+,=,=]	(59)
[A,=,#,=,=,=,=]	→	[C,=,A,=,+,=,=]	(60)
[C,=,−,=,=,=,=]	→	[C,=,−,=,+,=,=]	(61)
[C,=,+,=,=,=,=]	→	[C,=,=,=,+,=,=]	(62)
[+,*,+,*,#,*,*]	→	[−,#,*,#,#,#,#]	(63)
[A,#,+,=,*,#,#]	→	[*,#,A,=,−,#,#]	(64)
[B,#,+,=,*,#,#]	→	[*,#,B,=,−,#,#]	(64)
[C,#,+,=,*,#,#]	→	[*,#,C,=,−,#,#]	(64)
[A,#,#,=,*,#,+]	→	[*,#,#,=,−,#,A]	(65)
[B,#,#,=,*,#,+]	→	[*,#,#,=,−,#,B]	(65)
[C,#,#,=,*,#,+]	→	[*,#,#,=,−,#,C]	(65)
[A,#,#,=,*,+,#]	→	[*,#,#,=,−,A,#]	(66)
[B,#,#,=,*,+,#]	→	[*,#,#,=,−,B,#]	(66)
[C,#,#,=,*,+,#]	→	[*,#,#,=,−,C,#]	(66)
[*,+,+,+,#,#,#]	→	[#,*,*,*,#,#,#]	(67)
[#,=,+,=,*,#,#]	→	[+,−,=,−,#,#,#]	(68)
[#,=,#,=,*,#,+]	→	[−,B,#,C,#,#,=]	(69)
[C,=,+,=,*,=,=]	→	[+,−,=,−,#,−,−]	(82)
[−,*,−,*,#,*,*]	→	[#,#,−,#,#,#,#]	(70)
[*,#,+,−,*,#,#]	→	[A,#,A,=,#,#,#]	(71)
[*,#,+,B,#,#,*]	→	[B,#,A,=,#,#,#]	(72)
[*,#,+,C,#,*,#]	→	[C,#,A,=,#,#,#]	(73)
[*,#,#,−,*,#,+]	→	[B,#,#,B,#,#,A]	(83)
[*,#,#,−,*,+,#]	→	[C,#,#,C,#,A,#]	(84)
[A,#,+,A,#,#,−]	→	[A,#,A,+,#,#,−]	(85)
[A,#,+,B,#,#,−]	→	[B,#,A,+,#,#,−]	(85)
[A,#,+,C,#,#,−]	→	[C,#,A,+,#,#,−]	(85)
[A,#,+,A,#,−,#]	→	[A,#,A,+,#,−,#]	(86)
[A,#,+,B,#,−,#]	→	[B,#,A,+,#,−,#]	(86)
[A,#,+,C,#,−,#]	→	[C,#,A,+,#,−,#]	(86)
[+,=,+,=,*,#,#]	→	[+,A,=,A,#,#,#]	(74)
[−,=,#,=,*,#,+]	→	[+,B,#,C,#,#,=]	(75)
[+,=,+,=,*,=,=]	→	[+,A,=,A,#,A,A]	(87)
[A,#,+,A,−,#,#]	→	[*,=,A,+,#,=,=]	(76)
[B,#,+,B,#,#,−]	→	[*,=,B,+,#,=,=]	(77)
[C,#,+,C,#,−,#]	→	[*,=,C,+,#,=,=]	(78)
[B,#,#,A,−,#,+]	→	[*,=,=,B,=,#,B]	(88)
[C,#,#,A,−,+,#]	→	[*,=,=,C,=,C,#]	(89)
[A,A,+,#,#,A,#]	→	[A,+,A,#,#,+,#]	(90)
[B,A,+,#,#,A,#]	→	[A,+,B,#,#,+,#]	(90)
[C,A,+,#,#,A,#]	→	[A,+,C,#,#,+,#]	(90)
[+,=,+,=,#,#,#]	→	[#,−,=,−,#,#,#]	(79)
[+,C,+,C,#,−,−]	→	[C,C,=,C,#,A,A]	(91)
[+,B,+,B,#,−,−]	→	[B,B,=,B,#,A,A]	(91)
[*,#,+,−,#,#,#]	→	[+,=,*,+,#,=,=]	(80)
[*,#,#,C,#,#,+]	→	[+,=,=,C,=,#,*]	(92)
[*,#,#,B,#,#,+]	→	[+,=,=,B,=,#,*]	(92)
[#,+,+,+,#,#,#]	→	[*,*,*,*,#,#,#]	(81)
[C,C,+,C,#,+,+]	→	[C,*,=,*,#,=,=]	(93)
[B,B,+,B,#,+,+]	→	[B,*,=,*,#,=,=]	(93)
[B,#,#,+,#,*,#]	→	[*,#,#,B,#,−,#]	(94)
[C,#,#,+,#,*,#]	→	[*,#,#,C,#,−,#]	(94)
[*,#,#,+,#,*,#]	→	[A,#,#,A,#,−,#]	(95)
[C,#,+,#,#,=,=]	→	[C,+,=,+,#,=,=]	(96)
[B,#,+,#,#,=,=]	→	[B,+,=,+,#,=,=]	(96)
[#,#,#,+,#,#,−]	→	[#,#,#,−,#,#,=]	(97)
[A,#,#,+,#,−,#]	→	[*,#,=,A,=,*,=]	(98)
[*,#,#,=,#,#,+]	→	[+,=,=,−,=,#,*]	(99)
[#,#,#,+,#,#,*]	→	[#,#,#,+,#,#,*]	(100)
[C,−,+,−,#,=,=]	→	[C,C,=,C,#,=,=]	(101)
[B,−,+,−,#,=,=]	→	[B,B,=,B,#,=,=]	(101)
[#,#,#,C,#,#,−]	→	[#,#,#,C,#,#,*]	(102)
[#,#,#,B,#,#,−]	→	[#,#,#,B,#,#,*]	(102)
[C,+,+,+,#,=,=]	→	[C,#,=,#,#,B,B]	(103)
[B,+,+,+,#,=,=]	→	[B,#,=,#,#,C,C]	(103)
[C,=,+,=,#,C,C]	→	[+,B,=,B,#,#,#]	(104)
[B,=,+,=,#,B,B]	→	[+,C,=,C,#,#,#]	(104)
[*,*,#,=,B,*,*]	→	[−,#,#,B,+,#,#]	(105)
[*,=,#,=,B,#,#]	→	[B,B,#,B,−,#,#]	(106)
[−,*,#,B,A,*,*]	→	[A,#,−,A,+,−,−]	(107)
[−,*,#,B,B,*,*]	→	[B,#,−,B,+,#,−]	(108)
[−,B,#,*,C,*,*]	→	[C,C,−,#,+,#,−]	(109)
[B,B,#,B,−,#,#]	→	[B,*,#,*,+,#,#]	(110)
[A,#,#,*,A,#,#]	→	[A,#,*,A,+,*,*]	(111)
[B,#,#,*,A,#,#]	→	[A,#,*,B,+,#,*]	(112)
[C,*,#,#,A,#,#]	→	[A,C,*,#,+,#,*]	(113)
[B,A,#,A,−,#,#]	→	[B,=,A,=,+,A,A]	(114)
[B,B,#,C,−,#,#]	→	[B,=,A,=,+,#,A]	(115)
[B,A,#,A,=,#,#]	→	[#,=,A,=,+,A,A]	(116)
[B,B,#,C,=,#,#]	→	[#,=,A,=,+,#,A]	(117)
[A,#,+,=,*,#,+]	→	[*,#,A,=,−,#,A]	(118)
[A,=,+,#,*,#,+]	→	[*,=,A,#,−,#,A]	(119)
[A,#,+,=,*,+,+]	→	[*,#,A,=,−,A,A]	(120)
[#,=,+,=,*,#,+]	→	[−,B,=,C,#,#,=]	(121)
[#,+,+,+,*,+,+]	→	[−,=,=,=,#,=,=]	(122)
[*,#,+,B,*,#,+]	→	[B,#,B,=,#,#,B]	(123)
[*,C,+,#,*,#,+]	→	[C,=,B,#,#,#,B]	(124)
[*,#,+,=,*,+,+]	→	[A,#,B,=,#,B,B]	(125)
[−,=,+,=,*,#,+]	→	[+,B,=,C,#,#,=]	(126)
[−,=,+,=,*,+,+]	→	[+,−,=,−,#,=,=]	(127)
[A,#,+,−,−,+,+]	→	[*,=,A,+,#,C,B]	(128)
[B,#,+,B,−,#,+]	→	[*,=,B,+,#,#,B]	(129)
[C,C,+,#,−,#,+]	→	[*,+,C,=,#,#,C]	(130)
[+,=,+,=,#,#,+]	→	[#,−,=,−,#,#,=]	(131)
[+,=,+,=,#,+,+]	→	[#,−,=,−,#,=,=]	(132)
[*,#,+,−,#,#,+]	→	[+,=,*,+,#,#,*]	(133)
[*,#,+,−,#,+,+]	→	[+,=,*,+,#,*,*]	(134)
[#,+,+,+,#,#,+]	→	[*,*,*,*,#,#,*]	(135)
[#,+,+,+,#,+,+]	→	[*,*,*,*,#,*,*]	(136)
[+,*,+,*,#,#,+]	→	[−,#,*,#,#,#,*]	(137)
[+,*,+,*,#,+,+]	→	[−,#,*,#,#,*,*]	(138)
[*,+,+,+,#,#,+]	→	[#,*,*,*,#,#,*]	(139)
[*,+,+,+,#,+,+]	→	[#,*,*,*,#,*,*]	(140)
[−,*,−,*,#,#,−]	→	[#,#,−,#,#,#,−]	(141)
[−,*,−,*,#,−,−]	→	[#,#,−,#,#,−,−]	(142)
[A,#,+,=,D,#,#]	→	[D,#,A,=,+,#,#]	(143)
[B,#,+,=,D,#,#]	→	[D,#,B,=,+,#,#]	(143)
[C,#,+,=,D,#,#]	→	[D,#,C,=,+,#,#]	(143)
[A,#,#,=,D,#,+]	→	[D,#,#,=,+,#,A]	(144)
[B,#,#,=,D,#,+]	→	[D,#,#,=,+,#,B]	(144)
[C,#,#,=,D,#,+]	→	[D,#,#,=,+,#,C]	(144)
[A,#,#,=,D,+,#]	→	[D,#,#,=,+,A,#]	(145)
[B,#,#,=,D,+,#]	→	[D,#,#,=,+,B,#]	(145)
[C,#,#,=,D,+,#]	→	[D,#,#,=,+,C,#]	(145)
[A,#,+,#,D,#,#]	→	[D,#,A,#,+,#,#]	(146)
[B,#,+,#,D,#,#]	→	[D,#,B,#,+,#,#]	(146)
[C,#,+,#,D,#,#]	→	[D,#,C,#,+,#,#]	(146)
[A,#,#,#,D,#,+]	→	[D,#,#,#,+,#,A]	(147)
[B,#,#,#,D,#,+]	→	[D,#,#,#,+,#,B]	(147)
[C,#,#,#,D,#,+]	→	[D,#,#,#,+,#,C]	(147)
[D,+,#,A,#,#,=]	→	[A,D,#,+,#,#,=]	(148)
[D,+,#,#,A,#,=]	→	[A,D,#,#,+,#,=]	(149)
[D,+,A,#,#,#,=]	→	[A,D,+,#,#,#,=]	(150)
[D,+,#,#,A,D,#]	→	[*,C,*,#,+,−,#]	(151)
[D,+,#,#,A,#,D]	→	[*,C,*,#,+,#,−]	(152)
[#,A,+,#,#,*,*]	→	[D,+,=,A,#,=,=]	(153)

[*,A,+,+,#,#,=] → [*,+,A,A,#,#,=] (154)
[*,A,+,+,#,=,#] → [*,+,A,A,#,=,#] (155)
[−,A,+,+,#,#,=] → [*,+,A,B,#,#,=] (156)
[−,A,+,+,#,=,#] → [*,+,A,B,#,=,#] (157)
[+,A,+,+,#,#,=] → [*,+,A,C,#,#,=] (158)
[+,A,+,+,#,=,#] → [*,+,A,C,#,=,#] (159)
[*,B,+,+,#,#,=] → [−,+,A,A,#,#,=] (160)
[*,B,+,+,#,=,#] → [−,+,A,A,#,=,#] (161)
[−,B,+,+,#,#,=] → [−,+,A,B,#,#,=] (162)
[−,B,+,+,#,=,#] → [−,+,A,B,#,=,#] (163)
[+,B,+,+,#,#,=] → [−,+,A,C,#,#,=] (164)
[+,B,+,+,#,=,#] → [−,+,A,C,#,=,#] (165)
[*,C,+,+,#,#,=] → [+,+,A,A,#,#,=] (166)
[*,C,+,+,#,=,#] → [+,+,A,A,#,=,#] (167)
[−,C,+,+,#,#,=] → [+,+,A,B,#,#,=] (168)
[−,C,+,+,#,=,#] → [+,+,A,B,#,=,#] (169)
[+,C,+,+,#,#,=] → [+,+,A,C,#,#,=] (170)
[+,C,+,+,#,=,#] → [+,+,A,C,#,=,#] (171)
[D,=,+,#,#,−,−] → [#,+,=,A,#,=,=] (172)
[#,=,+,−,#,=,=] → [#,+,=,−,#,=,=] (173)
[D,C,#,+,#,=,#] → [+,+,#,A,#,=,#] (174)
[D,C,+,#,#,=,#] → [+,+,A,#,#,=,#] (175)
[D,C,#,#,+,=,#] → [+,+,#,#,A,=,#] (176)
[D,C,+,#,#,#,#] → [+,+,A,#,#,#,#] (177)
[+,A,#,+,#,=,#] → [−,+,#,A,#,=,#] (178)
[+,A,#,#,+,=,#] → [−,+,#,#,B,=,#] (179)
[+,A,+,#,#,=,#] → [−,+,C,#,#,=,#] (180)
[+,A,+,#,#,#,#] → [−,+,A,#,#,#,#] (181)
[−,A,#,+,#,=,#] → [C,+,#,D,#,=,#] (182)
[−,A,#,#,+,=,#] → [C,+,#,#,D,=,#] (183)
[−,A,+,#,#,=,#] → [C,+,D,#,#,=,#] (184)
[−,A,+,#,#,#,#] → [C,+,D,#,#,#,#] (185)
[*,D,+,+,#,=,#] → [D,+,A,A,#,*,#] (186)
[*,D,+,+,#,#,=] → [D,+,A,A,#,#,*] (187)
[−,D,+,+,#,=,#] → [D,+,A,B,#,*,#] (188)
[−,D,+,+,#,#,=] → [D,+,A,B,#,#,*] (189)
[+,D,+,+,#,=,#] → [D,+,A,C,#,*,#] (190)
[+,D,+,+,#,#,=] → [D,+,A,C,#,#,*] (191)
[D,C,+,+,#,=,#] → [D,+,A,*,#,−,#] (192)
[D,C,+,+,#,#,=] → [D,+,A,*,#,#,−] (193)

[#,=,+,+,#,*,*] → [D,+,=,*,#,=,=] (194)
[D,A,+,+,#,=,#] → [#,+,A,D,#,=,#] (195)
[D,A,+,+,#,#,=] → [#,+,A,D,#,#,=] (196)
[D,=,+,+,#,−,−] → [#,+,=,*,#,=,=] (197)
[#,A,+,−,#,=,#] → [A,+,A,−,#,=,#] (198)
[#,A,+,−,#,#,=] → [A,+,A,−,#,#,=] (199)
[*,#,+,#,D,#,−] → [A,#,D,#,#,#,=] (200)
[A,−,−,+,#,A,#] → [−,#,+,A,#,=,#] (201)
[A,−,−,+,#,#,A] → [−,#,+,A,#,#,=] (202)
[A,D,#,#,#,#,=] → [A,+,#,D,#,#,A] (203)
[−,+,#,#,D,−,#] → [D,D,−,#,+,*,#] (204)
[−,+,#,#,D,#,−] → [D,D,−,#,+,#,*] (205)
[#,#,A,+,#,=,=] → [#,#,+,=,#,D,D] (206)
[#,D,#,#,#,#,#] → [#,#,#,D,#,#,#] (207)
[−,+,#,#,*,−,#] → [A,A,B,#,+,=,#] (208)
[−,+,#,#,*,#,−] → [A,A,B,#,+,#,=] (209)
[A,+,#,#,A,#,D] → [*,A,#,#,+,#,=] (210)
[A,+,#,#,A,D,#] → [*,A,#,#,+,=,#] (211)
[#,B,#,#,#,#,#] → [#,B,#,#,#,#,#] (212)
[*,+,B,#,A,#,D] → [A,A,#,#,+,#,D] (213)
[*,+,B,#,A,D,#] → [A,A,#,#,+,D,#] (214)
[#,+,#,#,=,D,D] → [#,=,D,#,+,=,=] (215)
[#,+,#,D,=,=,=] → [#,=,#,B,+,D,D] (216)
[*,+,#,#,A,#,=] → [*,A,#,#,+,#,B] (217)
[*,+,#,#,A,=,#] → [*,A,#,#,+,B,#] (218)
[#,+,#,B,=,=,=] → [D,=,#,#,+,D,D] (219)
[*,+,#,#,A,#,D] → [D,D,−,#,+,#,−] (221)
[*,+,#,#,A,D,#] → [D,D,−,#,+,−,#] (222)
[D,+,#,#,=,B,B] → [B,=,#,#,+,D,D] (223)
[B,+,#,#,=,−,−] → [D,=,A,#,+,B,B] (224)
[*,+,+,#,A,#,B] → [*,A,A,#,+,#,B] (225)
[*,+,+,#,A,B,#] → [*,A,A,#,+,B,#] (226)
[#,+,−,#,=,B,B] → [#,=,−,#,+,B,B] (227)
[#,+,+,#,=,B,B] → [#,=,=,#,+,B,B] (228)
[*,+,+,#,B,#,B] → [−,A,A,#,+,#,B] (229)
[*,+,+,#,C,B,#] → [+,A,A,#,+,B,#] (230)
[D,C,+,+,#,B,#] → [#,+,A,*,#,−,#] (231)
[D,C,+,+,#,#,B] → [#,+,A,*,#,#,−] (232)
[#,+,+,#,A,#,=] → [#,A,*,#,+,#,=] (233)
[#,+,+,#,A,=,#] → [#,A,*,#,+,=,#] (234)

研究業績一覧

I 学会誌等学術研究論文

1. 牧川方昭, 今井克暢, 新土井賢, 田野岡和紀, 飯泉仁美, 三谷壽: マイクロコンピュータ技術を用いた生体信号無拘束計測装置, 計測自動制御学会論文集, **29**, (1993), 888–895.
2. Katsunobu Imai and Kenichi Morita: Firing squad synchronization problem in reversible cellular automata, *Theoretical Computer Science*, **165**, (1996), 475–482.
3. Kenichi Morita and Katsunobu Imai: Self-reproduction in a reversible cellular space. *Theoretical Computer Science*, **168**, (1996), 337–366.
4. 近宗克紀, 森田憲一, 今井克暢: 32 状態可逆セル・オートマトン上での論理回路合成, 電子情報通信学会論文誌, **J81-D-I**, (1998), 350–352.
5. Kenichi Morita, Satoshi Ueno and Katsunobu Imai: Characterizing the ability of parallel array generators on reversible partitioned cellular automata, *International Journal of Pattern Recognition and Artificial Intelligence* (to appear).
6. Kenichi Morita and Katsunobu Imai: Uniquely parsable array grammars for generating and parsing connected patterns, *Pattern Recognition Journal* (to appear).
7. Katsunobu Imai and Kenichi Morita: A computation-universal two-dimensional 8-state triangular reversible cellular automaton, *Theoretical Computer Science* (to appear).

II 国際会議発表研究論文

1. Kenichi Morita and Katsunobu Imai: Self-reproduction in a reversible cellular space, *Proceedings of International Workshop on Universal Machines and Computations*, Paris, (1995).
2. Katsunobu Imai, Taichi Adachi, Shinichi Furusaka and Kenichi Morita: Firing squad synchronization problem in one and two dimensional reversible cellular automaton, *Proceedings of Cellular Automata Workshop 96*, Gießen, (1996), 38–40.
3. Kenichi Morita and Katsunobu Imai: A simple self-reproducing cellular automaton with shape-encoding mechanism, *Artificial Life V*, The MIT Press, (1997), 489–496.
4. Kenichi Morita and Katsunobu Imai: Logical universality and self-reproduction in reversible cellular automata, *Evolvable Systems: From Biology to Hardware* (ICES96), LNCS1259, Springer, (1997), 152–166.
5. Kenichi Morita, Satoshi Ueno and Katsunobu Imai: Characterizing the ability of parallel array generators on reversible partitioned cellular automata, *Proceedings of International Workshop on parallel image analysis*, Hiroshima, (1997), 144-156.

6. Katsunobu Imai and Kenichi Morita: Generation and parsing of connected patterns by uniquely parsable array grammars, *Proceedings of International Workshop on parallel image analysis*, Hiroshima, (1997), 315–325.

7. Koji Okuhara, Katsunobu Imai, Kenichi Morita and Shunji Osaki: A design method for parameters of synergetic computers for image recognition, *Proceedings of International Workshop on parallel image analysis*, Hiroshima, (1997), 102–111.

8. Katsunobu Imai and Kenichi Morita: A computation-universal two-dimensional 8-state triangular reversible cellular automaton, *Proceedings of the Second Colloquium on Universal Machines and Computations*, Metz, (1998), 90–99.

9. Kenichi Morita, Maurice Margenstern, and Katsunobu Imai: Universality of reversible hexagonal cellular automata, *Proceedings of MFCS'98 Workshop on Frontiers between Decidability and Undecidability*, Brno, (1998).

10. Kenichi Morita and Katsunobu Imai: Number-Conserving Reversible Cellular Automata and Their Computation-Universality, *Proceedings of MFCS'98 Workshop on Cellular Automata*, Brno, (1998).

11. Katsunobu Imai, Kenichi Morita, and Kenji Sako: Firing Squad Synchronization Problem in Number-Conserving Cellular Automata, *Proceedings of AUTOMATA'98*, Santiago, (1998).

III 研究報告等

1. 牧川方昭, 今井勝喜, 新土井賢: 無拘束体調モニタ装置による身体活動，心活動の同時計測, 日本ME学会専門別研究会生体信号の長時間無拘束計測と解析研究会」研究報告集, 1-6, (1992), 211–216.

2. 今井勝喜, 牧川方昭, 佐藤俊輔: 種々の場面における心拍間隔，身体活動量の同時計測, 第6回生体・生理工学シンポジウム論文集 (1991), 431–434.

3. 安賀広幸, 今井克暢, 下條真司, 西尾章治郎, 宮原　秀夫: マルチメディアデータの扱いに適したオブジェクトモデル, オブジェクト指向コンピューティングI　日本ソフトウェア科学会WOOC'92, 近代科学社, (1993), 181–192.

4. 牧川方昭, 今井克暢, 新土井賢, 田野岡和紀, 飯泉仁美, 三谷壽: 種々の場面における心拍間隔，身体活動量の同時計測, *BME*, **7**, (1993), 48–54.

5. 安賀広幸, 今井克暢, 下條真司, 宮原秀夫: 柔軟な集合の扱いに向いたオブジェクトモデルの提案と応用第9回オブジェクト指向計算ワークショップ, (1993).

6. 今井克暢，森田憲一: 1次元可逆セル・オートマトンにおける一斉射撃問題について, 数理解析研究所講究録 *871*, (1994), 66–72.

7. 今井克暢, 森田憲一: 1次元可逆セル・オートマトンにおける一斉射撃問題について, 信学技報 COMP93-92, SS93-60, (1994), 41–48.

8. 今井克暢, 森田憲一: 1次元可逆セル・オートマトンにおける一斉射撃問題の高速解, 数理解析研究所講究録 *906*, (1995), 119–125.

9. 森田憲一, 今井克暢: 可逆セル空間における自己増殖, LA 夏のシンポジウム予稿集, (1995), 19–24.

10. 安達太一, 古阪真一, 今井勝喜, 森田憲一: 2次元可逆セル・オートマトンにおける一斉射撃問題, 数理解析研究所講究録 *950*, (1996), 228–232.

11. 今井克暢, 森田憲一: 計算万能な2次元可逆セル・オートマトンについて, LA 夏のシンポジウム予稿集, (1996), 19–24.

12. 今井克暢, 安達太一, 森田憲一: 2n 時間解に基づく1次元可逆セル・オートマトン上の一斉射撃解, LA 夏のシンポジウム予稿集, (1997), 87–92.

13. 今井克暢, 森田憲一: 計算万能な2次元8状態3角形状可逆セル・オートマトン, 数理解析研究所講究録 *992*, (1997), 228–232.

14. 森田憲一, 今井克暢: 形状符号化機構を持つ単純な自己増殖セル・オートマトン, LA 夏のシンポジウム予稿集, (1997), 93–98.

15. 今井克暢, 森田憲一: 計算万能な2次元8状態3角形状可逆セル・オートマトン, 信学技報 COMP97-80, (1998), 17–22.

16. 森田憲一, 今井克暢: Computation-Universality of One-Dimensional Number-Conseving Reversible Cellular Automata, LA 夏のシンポジウム予稿集, (1998), 113–118.

17. 今井克暢, 森田憲一: Firing Squad Synchronization Problem in Number-Conserving Cellular Automata, LA 夏のシンポジウム予稿集, (1998), 119–124.

主論文と引用既発表論文との対応

主論文	既発表論文
1. Introduction	
2. Cellular Automata	
3. Computation-Universal Two-Dimensional Reversible CA	I-7, II-4,8,9
4. Synchronization on Reversible and Conservative Cellular Automata	I-2, II-2,10,11
5. Self-Reproduction in Reversible Cellular Automata	I-3, II-1,3,4
6. Conclusion	