Proceedings of the International Workshop on Tilings and Cellular Automata, 2004

Maurice Margenstern
University of Metz, I.U.T. of Metz, France

CDMTCS-253
December 2004

Preface

This volume contains the papers sent to the workshop *Tilings and cellular automata* to be held in Auckland as a satellite workshop of **DLT'04**.

I am very thankful to the invited speakers, André Barbé, Jarkko Kari and Kenichi Morita for their very important contributions. I am also very thankful to the contributors who witness the wide and vivid activity of cellular automata in tight connection with tilings.

I wish also to express my thanks to Martin Kutrib, Hiroshi Umeo and Laurent Vuillon who helped me in the preparation of this workshop.

Last, but not at all the least, I am very thankful to Cristian S. Calude for inviting me to organise this workshop and for his help to present these proceedings.

Metz, November 24, 2004,

Maurice Margenstern

Invited Lectures

Coarse-graining invariant orbits of cellular automata

André Barbé

Dept. of Electrical Engineering, Katholieke Universiteit Leuven,
Kasteelpark Arenberg 10,
3001 Leuven, Belgium
e-mail: andre.barbe@esat.kuleuven.ac.be

November 23, 2004

Abstract

This talk presents an introduction to our work on properties of coarse-graining invariant CA-orbits. Coarse-graining is an operation that groups together cell-states in tiles that cover the orbit space.

1 Tiling and coarse-graining a CA's orbit

We start by considering a one-dimensional bi-infinite linear cellular automaton with states in a finite field $\mathbb{F}_q$ with $q = p^m$ elements, p being a prime. A configuration (or global state) of the CA is thus a sequence $(c(k))_{k\in\mathbb{Z}} \in \mathbb{F}_q^{\mathbb{Z}}$. The configuration at time $t \in \mathbb{N}$ will be denoted as $(c(t,k))_{k\in\mathbb{Z}}$. A configuration at time $t \in \mathbb{N}$ updates to a configuration at time $t+1$ according to a map

$$A : \mathbb{F}_q^{\mathbb{Z}} \to \mathbb{F}_q^{\mathbb{Z}} : (c(t,k))_{k\in\mathbb{Z}} \mapsto (c(t+1,k))_{k\in\mathbb{Z}}$$

which is defined by a *local evolution rule*

$$c(t+1,k) = \sum_{j\in\mathsf{N}} r(j)c(t,k-j) \qquad (\text{in } \mathbb{F}_q), \tag{1}$$

where $\mathsf{N} = \{a, a+1, \ldots, b-1, b\} \subset \mathbb{Z}$ defines a finite "neighbourhood"-set $k-\mathsf{N} = \{k-b, k-b+1, \ldots, k-a\}$ containing those cells whose states determine the next state of cell k, and where $r(j) \in \mathbb{F}_q$. The orbit of the cellular automaton is the two-dimensional sequence $O_A = (c(t,k))_{(t,k)\in\mathbb{N}\times\mathbb{Z}}$, representing the evolution of the CA-configuration over time. A specific orbit is completely determined by the initial configuration $(c(0,k))_{k\in\mathbb{Z}}$, and is graphically represented with cell-states at a given time displayed horizontally and with time pointing downwards. Figure 1 shows an example.

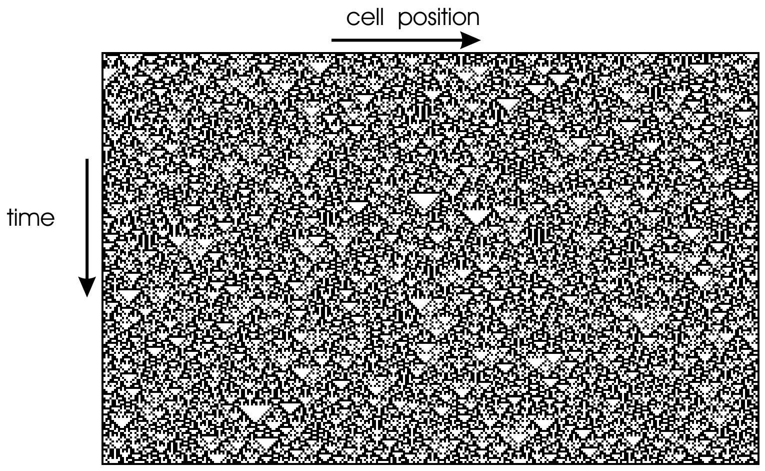


Figure 1: Part of an orbit of the CA with rule $c(t+1,k) = c(t,k-1)+c(t,k+2)$ in $\mathbb{F}_2 = \{0,1\}$ with addition and product modulo 2, and with random initial configuration (top row).

Now fix some $\sigma \in \mathbb{N}$ and $\tau \in \mathbb{Z}$ and define the corresponding q^n-decimation of the orbit, i.e., the sequence

$$D^{O_A}_{\sigma,\tau} = (d_{\sigma,\tau}(t,k))_{(t,k)\in\mathbb{N}\times\mathbb{Z}} = (c(q^n t + \sigma, q^n k + \tau))_{(t,k)\in\mathbb{N}\times\mathbb{Z}}. \tag{2}$$

Representing a sequence $(s(k))_{k\in\mathbb{Z}}$ as formal Laurent-series $S(X) = \sum_{k\in\mathbb{Z}} s(k)X^k$, and introducing the Laurent polynomial

$$R(X) = \sum_{j\in\mathsf{N}} r(j)X^j, \tag{3}$$

the above convolution (1) can be written as

$$C(t+1,X) = R(X)C(t,X). \tag{4}$$

Iterating this for s time steps gives the configuration at time $t+s$ in dependency on the configuration at time t:

$$C(t+s,X) = (R(X))^s C(t,X). \tag{5}$$

In the sequence domain, this corresponds to

$$c(t+s,k) = \sum_{j\in\mathsf{N}_s} r_s(j)c(t,k-j), \tag{6}$$

where $r_s(j)$ are the coefficients in the polynomial $(R(X))^s = \sum_{j\in\mathsf{N}_s} r_s(j)X^j$. Taking $s = q^n$ in the above formulas gives the configuration at time $t+q^n$ in its dependency on the configuration at time t:

$$C(t+q^n,X) = (R(X))^{q^n} C(t,X). \tag{7}$$

Now we can use a well-known property of polynomials in $\mathbb{F}_q$, namely that $(R(x))^{q^n} = R(X^{q^n})$. Thus, (7) becomes

$$C(t + q^n, X) = R(X^{q^n})C(t, X). \tag{8}$$

Reinterpreting this in terms of sequences, and taking into account that the coefficient in $R(X^{q^n})$ corresponding to X^k equals 0 when k is not a multiple of q^n, and equals $r(j)$ when $k = jq^n$, establishes the following relation between the configurations $(c(t + q^n, k))_{k\in\mathbb{Z}}$ and $(c(t, k))_{k\in\mathbb{Z}}$

$$c(t + q^n, k) = \sum_{j\in\mathsf{N}} r(j)c(t, k - q^n j) \tag{9}$$

Now reconsider the decimation sequence as defined in (2). We have

$$d_{\sigma,\tau}(t, k) \overset{(2)}{=} c(\sigma + q^n t, \tau + q^n k) \overset{(9)}{=} \sum_{j\in\mathsf{N}} r(j)c(\sigma + q^n(t - 1), \tau + q^n(k - j))$$

Invoking (2) again, this shows that

$$d_{\sigma,\tau}(t, k) = \sum_{j\in\mathsf{N}} r(j)d_{\sigma,\tau}(t - 1, k - j), \tag{10}$$

meaning that the q^n-decimation $D^{O_A}_{\sigma,\tau}$ of an orbit is an orbit of the same CA, i.e. it satisfies the same local rule as specified in (1).

But there is even more. Define a basic $(q^n \times q^n)$-tile $T_{0,0}$, as the set of $q^n \times q^n$ orbit cells, specified by their coordinates, as given by

$$T_{0,0} = \{(i, c_j)|i = 0, 1, \ldots, q^n - 1, c_j = j + \gamma_{i,j}q^n + \beta_i + \delta,\ j \in \{0, 1, \ldots, q^n - 1\}\},$$

with each of the $\gamma_{i,j}, \beta_i$ and δ fixed integers. Figure 2 displays some 4×4-tiles. Then the (κ, ν)-shifted tiles defined by

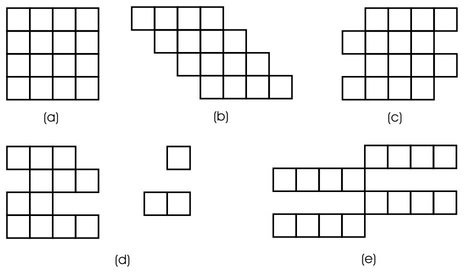


Figure 2: Examples of 4×4-tiles. As (d)shows, tiles may be unconnected.

$$T_{\kappa,\nu} = T_{0,0} + (\kappa q^n, \nu q^n)$$

for $(\kappa, \nu) \in \mathbb{N} \times \mathbb{Z}$ are all disjoint and form a tiling of the orbit-support $\mathbb{N} \times \mathbb{Z}$. Let this tiling be denoted by

$$\mathcal{T}_0 = \{T_{\kappa,\nu} | (\kappa, \nu) \in \mathbb{N} \times \mathbb{Z}\}.$$

Then consider the coarse-graining operation on the orbit that aggregates the states in each tile, by defining a new state $S(\kappa, \nu)$ on tile $T_{\kappa,\nu}$, as follows

$$S(\kappa, \nu) = \sum_{(t,k) \in T_{0,0}} \zeta(t, k) c(t + \kappa q^n, k + \nu q^n) = \sum_{(t,k) \in T_{0,0}} \zeta(t, k) d_{t,k}(\kappa, \nu), \quad (11)$$

where $\zeta(t, k) \in \mathbb{F}_q$ weights the contribution of the cell-state at relative position (t, c) in the tile.

Then (10) and (11) together imply that

$$S(\kappa, \nu) = \sum_{j \in \mathsf{N}} c(j) S(\kappa - 1, \nu - j).$$

I.e., $(S(\kappa, \nu))_{(\kappa,\nu) \in \mathbb{N} \times \mathbb{Z}}$ is also an orbit of the same cellular automaton.

Now there are q^n distinct tilings, namely

$$\mathcal{T}_\theta = \mathcal{T}_0 + (0, \theta) = \{T_{\kappa,\nu} + (0, \theta) | (\kappa, \nu) \in \mathbb{N} \times \mathbb{Z}\}, \quad \theta = 0, 1, \ldots, q^n - 1,$$

and one can consider the same coarse-graining of the CA-orbit as defined in (11) on all these tilings, i.e.

$$S^\theta(\kappa, \nu) = \sum_{(t,k) \in T_{0,0}} \zeta(t, k) c(t + \kappa q^n, k + \theta + \nu q^n), \quad (12)$$

each one resulting in a new orbit that satisfies

$$S^\theta(\kappa, \nu) = \sum_{j \in \mathsf{N}} c(j) S^\theta(\kappa - 1, \nu - j).$$

2 Orbits that are invariant under coarse-graining

The fact that the above defined coarse-grainings of a CA-orbit preserve the property of being an orbit for the same CA, evokes the question whether there exist orbits which are invariant under these coarse-grainings. I.e., do there exist orbits such that for all $\theta \in \{0, 1, \ldots, q^n - 1\}$, it holds that

$$S^\theta(\kappa, \nu) = c(\kappa, \nu), \quad (13)$$

for all $(\kappa, \nu) \in \mathbb{N} \times \mathbb{Z}$. Or, even more general, if we do not distinguish between orbits $c_1(t, k)$ and $c_2(t, k)$ if $c_1(t, k) = c_2(t, k + \alpha)$ for some $\alpha \in \mathbb{Z}$, then (13) may be generalized into

$$S^\theta(\kappa, \nu) = c(\kappa, \nu + \alpha_\theta) \text{ for all } \theta \in \{0, 1, , \ldots, q^n - 1\}. \quad (14)$$

Finding coarse-graining invariant (CGI) orbits thus requires solving (14). Because both $S^\theta(\kappa, \nu)$ and $c(\kappa, \nu)$ are orbits of the same CA, condition (14) can already be restricted to the initial configurations (time 0) as the rest of the orbit just follows by applying the CA's local evolution rule. This restriction reads:

$$S^\theta(0, \nu) = c(0, \nu + \alpha_\theta) \text{ for all } \theta \in \{0, 1, , \ldots, q^n - 1\} \text{ and } \nu \in \mathbb{Z}. \tag{15}$$

Now,

$$S^\theta(0, \nu) = \sum_{(t,k)\in T_{0,0}} \zeta(t, k) c(t, k + \theta + \nu q^n) \tag{16}$$

and, using (6), this eventually reduces to the form

$$S^\theta(0, \nu) = \sum_{j\in \mathsf{Z}} \eta(j) c(0, \nu q^n + j + \theta) \tag{17}$$

where Z is some finite subset of $\mathbb{Z}$ which, like the $\eta(j)$, depends on the CA-rule and the tiling-parameters.

Thus, the *coarse-graining invariance equations* (15) become

$$\sum_{j\in \mathsf{Z}} \eta(j) c(0, \nu q^n + j + \theta) = c(0, \nu + \alpha_\theta) \text{ for all } \theta \in \{0, 1, , \ldots, q^n - 1\} \text{ and } \nu \in \mathbb{Z}. \tag{18}$$

This is a set of difference equations in the initial configuration $(c(0, k))_{k\in\mathbb{Z}}$, featuring rescaled arguments.

We will show, by means of an example, how these equations can be solved. Notice that there is always a trivial solution, namely $c(0, k) = 0$ for all $k \in \mathbb{Z}$, leading to an orbit which is overall 0.

Example. It concerns a CA with values in $\mathbb{F}_2 = \{0, 1\}$ with addition and product taken modulo 2. Let the local evolution rule be given by

$$c(t + 1, k) = c(t, k - 1) + c(t, k + 1), \tag{19}$$

what corresponds to the polynomial form $R(X) = X^{-1}+X$. Consider the $q\times q = 2 \times 2$ tiling (i.e. $q = 2^1$) with basic tile $T_{0,0} = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$(i.e., $\gamma(i, j) = 0, \beta_j = 0$ for $i, j \in \{0, 1\}$). For the coarse-graining, the states of the cells in a tile are added with the same weight, i.e. $\zeta(t, k) = 1$ for $(t, k) \in T_{0,0}$. The α-shifts are $\alpha_0 = 0, \alpha_1 = 2$. The tilings $\mathcal{T}_0$ and $\mathcal{T}_1$ are represented in Figure 3.

With these parameters, the coarse-graining states (16) become

$$\begin{aligned} S^0(0, \nu) &= c(0, 2\nu) + c(0, 2\nu + 1) + c(1, 2\nu) + c(1, 2\nu + 1) \\ S^1(0, \nu) &= c(0, 2\nu + 1) + c(0, 2\nu + 2) + c(1, 2\nu + 1) + c(1, 2\nu + 2) \end{aligned} \tag{20}$$

which, using (19) for $t = 0$, and writing $c(k)$ instead of $c(0, k)$, reduces to the form of equation (17)

$$\begin{aligned} S^0(0, \nu) &= c(2\nu) + c(2\nu + 1) + c(2\nu - 1) + c(2\nu + 1) + c(2\nu) + c(2\nu + 2) \\ &= c(2\nu - 1) + c(2\nu + 2) \\ S^1(0, \nu) &= c(2\nu) + c(2\nu + 3). \end{aligned} \tag{21}$$

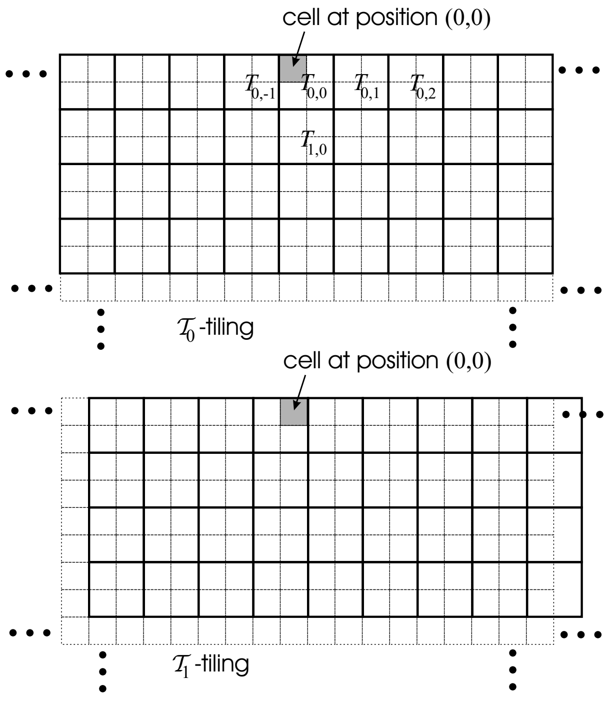


Figure 3: The tilings $\mathcal{T}_0$ and $\mathcal{T}_1$ of the Example. The small thin squares represent the orbit-cells, the fat squares represent the tiles. The coarse-graining under consideration adds the states of all cells in a tile.

With $\alpha_0 = 0$ and $\alpha_1 = 2$, the coarse-graining invariance equations (18) thus become

$$\begin{aligned} c(2\nu-1)+c(2\nu+2) &= c(\nu) \\ c(2\nu)+c(2\nu+3) &= c(\nu+2), \end{aligned} \tag{22}$$

for all $\nu \in \mathbb{Z}$.

These equations can be rewritten in a so-called left-propagating way:

$$\begin{aligned} c(2\nu-1) &= c(2\nu+2)+c(\nu) \\ c(2\nu) &= c(2\nu+3)+c(\nu+2), \end{aligned} \tag{23}$$

and also in a right propagating way:

$$\begin{aligned} c(2\nu+2) &= c(2\nu-1)+c(\nu) \\ c(2\nu+3) &= c(2\nu)+c(\nu+2), \end{aligned} \tag{24}$$

Inspection shows that, if the values $c(-1), c(0), c(1)$ are known, then the left-propagating form can be used to "grow" the solution $c(k)$ for $k < -1$, i.e. to the left, while the right propagating form can be used to determine $c(k)$ for $k > 1$.

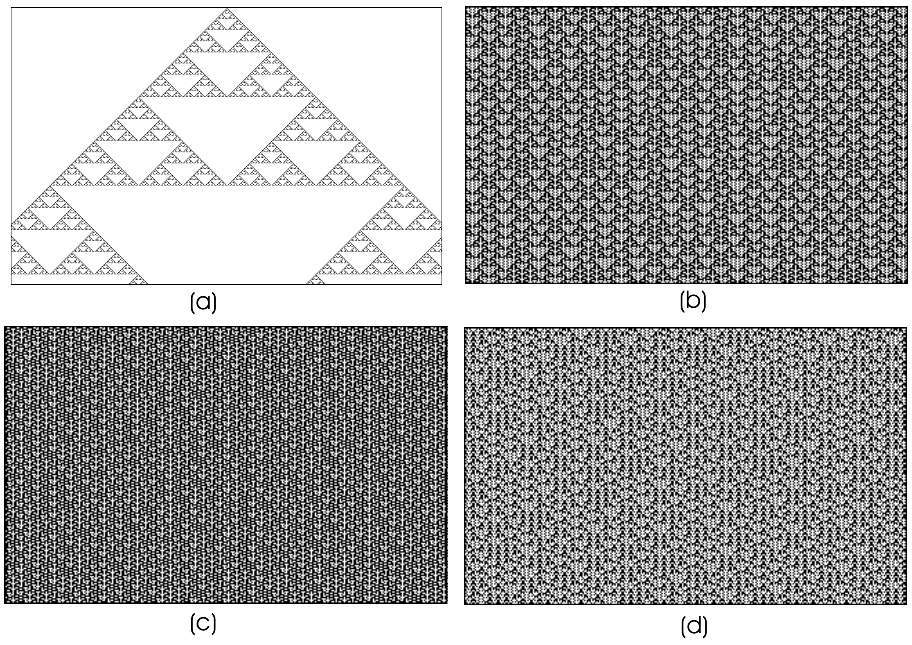


Figure 4: (a), (b), (c): three of the eight coarse-graining invariant solutions for the Example. The remaining solutions are similar to (b) and (c) (without being identical). Solution (a) is self-similar, solutions (b) and (c) are kind of quasi-periodic. (d) displays the addition modulo 2 of orbit(c) and a shifted version itself (here a horizontal shift over 1 cell).

So there are eight solutions, as the values $c(-1), c(0), c(1)$ can be freely chosen to be either 0 or 1.

Figure 4 displays a few of the solutions. These solutions show quasi-periodic properties, and display a hidden structure that appears when the image is added modulo 2 with a shifted version of itself (see Figure 5).

The procedure followed in the above example for solving the CGI-equations is generic: it is always possible to determine an interval $[kmin, kmax]$ such that knowledge of the initial configuration in this interval allows to propagate the whole solution. Only, it may be possible that the values in this interval are not independent (this is the case when all arguments in some of the CGI-equations lie inside this interval: the dependencies are of course given by the related CGI-equation [1].

In general, coarse-graining invariant orbits display interesting complexities, which range from total order (periodicity) to complete disorder (randomness), over quasi-periodic, locally periodic, self-similar, and quasi-random structures. The quasi-periodic and quasi-random structures have surprising hidden features which appear by comparing these structures with shifted versions of themselves.

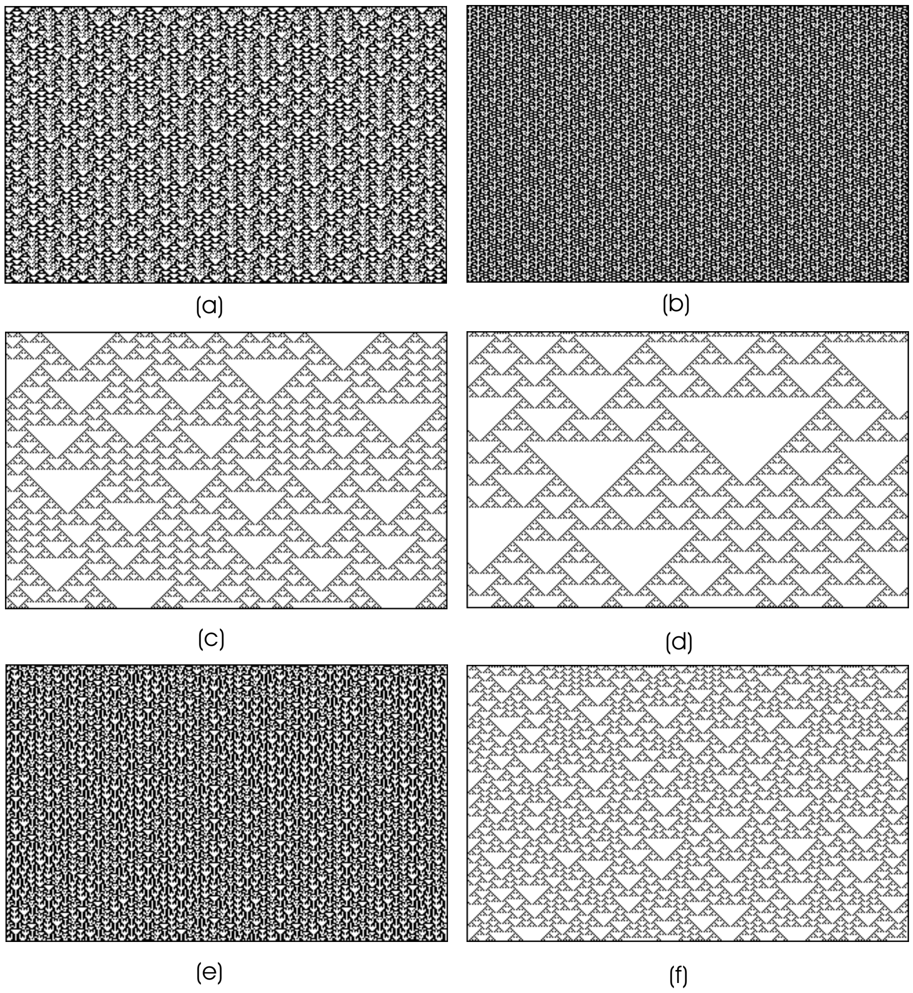


Figure 5: Further additions modulo 2 of orbit (c) from the previous Figure and its (h, v)- shifted versions (h denotes the horizontal shift, v the vertical shift). (a): $(h, v) = (5, 0)$; (b):$(h, v) = (4, 0)$; (c)$(h, v) = (24, 0)$; (d)$(h, v) = (24, 8)$; (e):$(h, v) = (24, 7)$; (f)$(h, v) = (24, 20)$.

See [2] for additional examples.

Another property of these two-dimensional coarse-graining invariant orbits is that they are p-automatic sequences (whereby the state of a cell in the orbit can be generated by a finite automaton whose input is the p-adic representation of the cell's position)[3].

Similar three-dimensional coarse-graining invariant orbits also exist for two-dimensional CA. But in this situation, there are generally an infinite number of solutions (except for the coarse-graining which reduces to a pure decimation), most of which are not automatic any longer [4].

Coarse-graining invariant orbits of CA are special cases of coarse-graining invariant sequences that satisfy equations like (18), but without being necessarily related to cellular automata. The special case of decimation invariance, whereby coarse-graining reduces to picking just one cell in each tile (relative position of the cell in each tile the same) has been considered in [5], [6].

References

[1] A. Barbé, F. von Haeseler, H.-O. Peitgen, G. Skordev, Coarse-graining invariant patterns of one-dimensional two-state linear cellular automata, International Journal of Bifurcation and Chaos, Vol.5, no.6, 1995, 1611-1631.

[2] A. Barbé, Coarse-graining invariant orbits of one-dimensional $\mathbb{Z}_p$ linear cellular automata, International Journal of Bifurcation and Chaos, Vol.6, no.12A, 1996, 2237-2297.

[3] A. Barbé, Complex order from disorder and from simple order in coarse-graining invariant orbits of certain two-dimensional linear cellular automata, International Journal of Bifurcation and Chaos, Vol.7, no.7, 1997, 1451-1496.

[4] A. Barbé, H.-O.Peitgen, G. Skordev, Automaticity of coarse-graining invariant orbits of one-dimensional linear cellular automata, International Journal of Bifurcation and Chaos, Vol.9, no.1, 1999, 67-95.

[5] A. Barbé, G. Skordev, Decimation-invariant sequences and their automaticity, Theoretical Computer Science, 259, 2001, 379-403.

[6] A. Barbé, F. von Haeseler, genralized decimation-invariant sequences in dimension N, International Journal of Bifurcation and Chaos, Vol.12, no.4, 2002, 709-737.

The tiling problem and undecidability in Cellular Automata

Jarkko Kari*

Department of Mathematics
FIN-20014 University of Turku, Finland
e-mail: jkari@utu.fi

November 23, 2004

Abstract

Many questions concerning one- and two-dimensional cellular automata can be proved undecidable via a reduction from the tiling problem of Wang tiles. Examples include nilpotency (in 1D and 2D), reversibility (in 2D) and surjectivity (in 2D) problems of CA. We review the basic ideas of these reductions. We also discuss variants of the tiling problem that, in turn, can be proved undecidable using similar constructions.

1 Introduction

Berger's classic result on the undecidability of the tiling problem is the basis of many undecidability proofs concerning cellular automata (CA). Variants of the tiling problem have been used in reductions to prove, for example, that there are no algorithms to determine if a given two-dimensional CA is reversible or surjective [10, 13], or whether a given one-dimensional CA is nilpotent [12]. Also there is no algorithm to compute the topological entropy of a CA [9]. The ideas developed for the undecidability proofs concerning CA have also contributed to tiling problems. One particular example is the snake tiling problem [1, 16]. In the following we review the basic ideas of the reductions. We start by briefly recalling the definitions and some basic results on cellular automata and Wang tiles.

1.1 Cellular Automata

In this paper we consider synchronous cellular automata only, where the underlying topology is an infinite rectangular grid. The cells are hence the squares of

*Research supported by the Academy of Finland grant 54102

an infinite d-dimensional checker board, addressed by $\mathbb{Z}^d$. We are in particular interested in the one- and two-dimensional cases.

The states of the automaton come from a finite *state set* S. At any given time, the *configuration* of the automaton is a mapping $c : \mathbb{Z}^d \longrightarrow S$ that specifies the states of all cells. The set $S^{\mathbb{Z}^d}$ of all configurations is denoted by $\mathcal{C}(d, S)$, or briefly $\mathcal{C}$ when d and S are known from the context. Constant functions are called *homogeneous* configurations.

The cells change their states synchronously at discrete time steps. The next state of each cell depends on the current states of the neighboring cells according to an update rule. All cells use the same rule, and the rule is applied to all cells at the same time. The neighboring cells may be the nearest cells surrounding the cell, but more general neighborhoods can be specified by giving the relative offsets of the neighbors. Let $N = (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n)$ be a vector of n distinct elements of $\mathbb{Z}^d$. Then the *neighbors* of a cell at location $\vec{x} \in \mathbb{Z}^d$ are the n cells at locations

$$\vec{x} + \vec{x}_i, \text{ for } i = 1, 2, \ldots, n.$$

The *local rule* is a function $f : S^n \longrightarrow S$ where n is the size of the neighborhood. State $f(a_1, a_2, \ldots, a_n)$ is the state of a cell whose n neighbors were at states $a_1, a_2, \ldots, a_n$ one time step before. This update rule then determines the global dynamics of the CA: Configuration c becomes in one time step the configuration e where, for all $\vec{x} \in \mathbb{Z}^d$,

$$e(\vec{x}) = f(c(\vec{x} + \vec{x}_1), c(\vec{x} + \vec{x}_2), \ldots, c(\vec{x} + \vec{x}_n)).$$

We say that $e = G(c)$, and call $G : \mathcal{C} \longrightarrow \mathcal{C}$ the *global transition function* of the CA.

In summary, cellular automata are dynamical systems that are homogeneous and discrete in both time and space, and that are updated locally in space. A d-dimensional CA is specified by a triple (S, N, f) where S is the state set, $N \in (S^{\mathbb{Z}^d})^n$ is the neighborhood vector, and $f : S^n \longrightarrow S$ is the local update rule. We usually identify a cellular automaton with its global transition function G, and talk about cellular automaton function G, or simply cellular automaton G. In algorithmic questions G is however always specified using the three finite items S, N and f.

In the one-dimensional case $d = 1$ a radius-r CA uses the neighborhood $(-r, -r + 1, \ldots, r - 1, r)$ of size $2r + 1$. A one-dimensional CA that uses the neighborhood $(0, 1)$ is sometimes called a radius-$\frac{1}{2}$ CA. This neighborhood is not symmetric, and no information can flow to the positive direction, but Figure 1 shows how the neighborhood can be made to look symmetric by shifting the cells to the right.

The *shift functions* are particularly simple CA that translate the configurations one cell down in one of the coordinate directions. More precisely, for each dimension $i = 1, 2, \ldots, d$ there is the corresponding shift function σ_i whose neighborhood contains only the unit coordinate vector $\vec{e}_i$ and whose local rule is the identity function. The one-dimensional shift function is the left shift $\sigma = \sigma_1$.

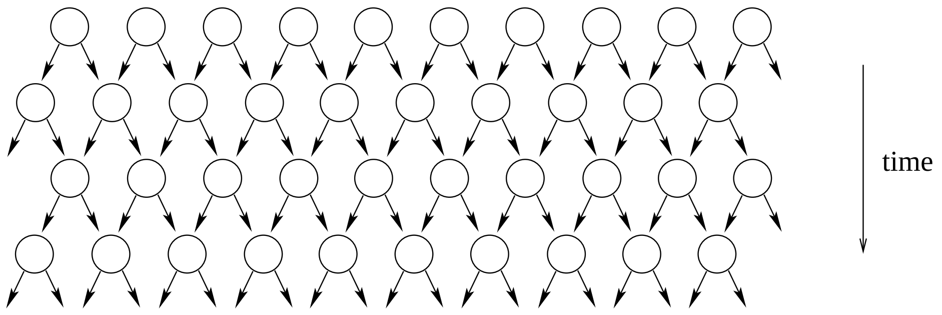


Figure 1: Dependencies in one-dimensional, radius-$\frac{1}{2}$ cellular automata.

Translations are compositions of shift functions. Translation $\tau_{\vec{y}}$ by vector $\vec{y}$ is the CA with neighborhood $(\vec{y})$ and the identity local rule.

Sometimes one state $q \in S$ is specified as a *quiescent state*. It should be *stable*, which means that $f(q, q, \ldots, q) = q$. The quiescent configuration Q is the configuration where all cells are quiescent: $Q(\vec{x}) = q$ for all $\vec{x} \in \mathbb{Z}^d$. A configuration $c \in S^{\mathbb{Z}^d}$ is called *finite* if only a finite number of cells are non-quiescent, i.e. the support

$$\{\vec{x} \in S^{\mathbb{Z}} \mid c(\vec{x}) \neq q\}$$

is finite. Let us denote by $\mathcal{C}_F(d, S)$, or briefly $\mathcal{C}_F$, the subset of $S^{\mathbb{Z}^d}$ that contains only the finite configurations. Because of stability of q, finite configurations remain finite in the evolution of the CA, so the restriction G_F of G on the finite configurations is a function $\mathcal{C}_F \longrightarrow \mathcal{C}_F$.

A *periodic configuration*, or more precisely, a spatially periodic configuration is a configuration that is invariant under d linearly independent translations. This is equivalent to the existence of d positive integers $t_1, t_2, \ldots, t_d$ such that $c = \sigma_i^{t_i}(c)$ for every $i = 1, 2, \ldots, d$, that is,

$$c(\vec{x}) = c(\vec{x} + t_i \vec{e}_i),$$

for every $\vec{x} \in \mathbb{Z}^d$ and every $i = 1, 2, \ldots, d$. Let us denote by $\mathcal{C}_P(d, S)$, or briefly $\mathcal{C}_P$, the set of periodic configurations. Cellular automata are homogeneous in space and consequently they preserve periodicity of configurations. The restriction G_P of G on the periodic configurations is hence a function $\mathcal{C}_P \longrightarrow \mathcal{C}_P$.

One must take care not to confuse (spatially) periodic configurations with temporally periodic configurations. Configuration c is *temporally periodic* for cellular automaton G if $G^k(c) = c$ for some $k \geq 1$. If $G(c) = c$ then c is a *fixed point*. Every cellular automaton has a temporally periodic configuration that is homogeneous: this follows from the facts that there are finitely many homogeneous configurations and that cellular automata functions preserve homogeneity. Configuration c is *eventually periodic* if it evolves into a temporally periodic configuration, that is, if $G^m(c) = G^n(c)$ for some $m \neq n$. This is equivalent to the property that the *(forward) orbit* $c, G(c), G^2(c), \ldots$ is finite. Every spatially periodic configuration is eventually periodic.

Cellular automaton G is called *nilpotent* if $G^n(\mathcal{C})$ is a singleton set for sufficiently large n, that is, if there is a configuration c and number n such that $G^n(e) = c$ for all configurations e. Since homogeneous configurations remain homogeneous we immediately see that configuration c has to be homogeneous and a fixed point.

Let G_1 and G_2 be two given cellular automata functions with the same state set and the same dimension d. The *composition* $G_1 \circ G_2$ is also a cellular automaton function, and the composition can be formed effectively. If N_1 and N_2 are neighborhoods of G_1 and G_2, respectively, then a neighborhood of $G_1 \circ G_2$ consists of vectors $\vec{x} + \vec{y}$ for all $\vec{x} \in N_1$ and $\vec{y} \in N_2$.

The equivalence of two given cellular automata G_1 and G_2 is decidable: If the neighborhoods N_1 and N_2 are the same then the local rules must be identical, and if the neighborhoods are different then one can take the union of the two neighborhoods and test whether the two cellular automata agree on the expanded neighborhood.

Sometimes it happens that $G_1 \circ G_2 = G_2 \circ G_1 = id$ where id is the identity function. Then cellular automata G_1 and G_2 are called *reversible* and G_1 and G_2 are the *inverse automata* of each other. One can effectively decide whether two given cellular automata are inverses of each other. This follows from the effectiveness of the composition and the decidability of the cellular automata equivalence.

1.2 Cantor topology

A seminal paper in the topological investigation of cellular automata is by Hedlund [8]. This paper is remarkable in several ways. It marks the beginning of symbolic dynamics, the study of bi-infinite words and the shift function.

Let us define a topology on the configuration space $S^{\mathbb{Z}^d}$. The topology we use is the Cantor topology, obtained as the infinite power of the discrete topological space S. This topology is compact by Tychonoff's theorem. A useful basis of the topology consists of the *cylinder sets*. Radius r cylinder containing configuration c is the set

$$\mathrm{Cyl}(c, r) = \{ e \in S^{\mathbb{Z}^d} \mid e(\vec{x}) = c(\vec{x}) \text{ whenever } ||\vec{x}||_\infty \leq r \ \}$$

where

$$||(x_1, x_2, \ldots, x_d)||_\infty = \max\{|x_1|, |x_2|, \ldots, |x_d|\}$$

is the max-norm. In other words, $\mathrm{Cyl}(c, r)$ consists of those configurations that agree with c at all cells whose coordinates are within distance r from 0. For every fixed r there are finitely many radius r cylinders, and these cylinders are disjoint. Hence the radius r cylinders partition $S^{\mathbb{Z}^d}$. It follows that each cylinder is *clopen*, that is, both open and closed in the topology. The complement of a radius r cylinder is namely the union of the other radius r cylinders.

It is easy to see that cellular automata functions are continuous in this topology. Trivially they commute with the shift functions σ_j, that is,

$$\sigma_j \circ G = G \circ \sigma_j$$

for every cellular automaton G and $j = 1, 2, \ldots, d$. The converse also holds. This is the Curtis-Hedlund-Lyndon theorem:

Theorem 1.1 ([8]) *A function $G : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$ is the global transition function of a cellular automaton if and only if*

(i) G is continuous, and

(ii) G commutes with the shifts σ_j.

If G is a reversible CA function then $G : \mathcal{C} \longrightarrow \mathcal{C}$ is by definition a bijection. Conversely, every CA function G that is bijective is reversible. Indeed, it's inverse function clearly commutes with the shift. The inverse function is also continuous because the space $\mathcal{C}$ is compact, and therefore it is a cellular automaton function. We have

Corollary 1.1 ([8]) *A cellular automaton G is reversible if and only if it is a bijection.*

1.3 Wang tiles

Wang tiles were introduced by logician H. Wang in 1961 [25]. They are relevant to cellular automata theory for several reasons. Some decision problems concerning cellular automata can be formulated as tiling problems, and the famous undecidability results concerning Wang tiles can then be employed to establish undecidability results in cellular automata. Aperiodic tiles can be used to provide interesting examples of two-dimensional cellular automata. Wang tiles are also used in one-dimensional CA where space-time diagrams can be viewed as tilings and this provides insight to the dynamics of the CA.

A *Wang tile* t is a unit square with colored edges. Let us denote by t_N, t_E, t_S and t_W the colors of the north, east, south and west edges of tile t, respectively. A *tile set* T is a finite collection of Wang tiles. A *Wang tiling* with T is a mapping $t : \mathbb{Z}^2 \longrightarrow T$, that is, copies of tiles in T are placed at integer lattice points, without rotating or flipping the tiles. Tiling t is valid at point $(x, y) \in \mathbb{Z}^2$ if the the colors of tile $t(x, y)$ match with the colors of the neighboring tiles,that is, if

$$\begin{array}{rcl} t(x,y)_N & = & t(x,y+1)_S, \\ t(x,y)_S & = & t(x,y-1)_N, \\ t(x,y)_E & = & t(x+1,y)_W, \text{ and} \\ t(x,y)_W & = & t(x-1,y)_E. \end{array}$$

Tiling t is *valid* if it is valid at every point $(x, y) \in \mathbb{Z}^2$. We say that tile set T admits a valid tiling if at least one valid tiling exists.

The set $T^{\mathbb{Z}^2}$ is given the compact Cantor topology discussed in the previous section. Then the set of valid tilings forms a compact subset of $T^{\mathbb{Z}^2}$. Moreover, if T admits valid tilings of arbitrarily large squares then it admits a valid tiling of the entire infinite plane.

The tiling question is the decision problem to determine if a given tile set T admits at least one valid tiling. The question was proved undecidable by R.Berger in 1966:

Theorem 1.2 ([3, 22]) *It is undecidable whether a given finite tile set T admits a valid tiling.*

Analogously to configurations, a tiling t is called *periodic* if it is invariant under two non-parallel translations. These translation can be chosen horizontal and vertical, which means that a periodic tiling consists of horizontal and vertical repetitions of a tiled rectangle. A tile set T that admits valid tilings is called *aperiodic* if it does not admit a valid periodic tiling. Already H. Wang observed [25] that if no aperiodic tile sets existed then the tiling problem would be decidable. One could namely try to tile larger and larger rectangles until one of the following two things happens: A rectangle is found that cannot be tiled, or a period of a periodic tiling is found. In the first case the tile set does not admit a tiling, in the second case it does. Only aperiodic tile sets fail to halt. So, as a corollary to Theorem 1.2, we have

Corollary 1.2 *Aperiodic tile sets exist.*

In fact, Berger's proof of Theorem 1.2 contains a construction of one aperiodic tile set. Currently, the smallest aperiodic set of Wang tiles contains 13 tiles [5, 15], and it is an open problem whether one of the tiles in this set is in fact superfluous.

When space-time diagrams of one-dimensional, radius-$\frac{1}{2}$ cellular automata are described using Wang tiles it turns out to be natural to consider tile sets in which the colors of two edges uniquely determine each tile. We call tile set T *NW-deterministic* if for every $s, t \in T$ we have

$$\left.\begin{array}{l} s_N = t_N \\ s_W = t_W \end{array}\right\} \Longrightarrow s = t.$$

This means that in a valid tiling each tile is uniquely determined by its neighbors to the north and to the west. The idea of NW-tile sets is that any valid tiling can be viewed as a space-time diagram of a cellular automaton with state set T, where the configurations are read along the infinite SW/NE-diagonals of the tiling.

The following observations were made in [12]:

Theorem 1.3 ([12]) *The tiling problem is undecidable when restricted to NW-deterministic tile sets. There are NW-deterministic, aperiodic tile sets.*

An other undecidable variant of the tiling problem is the *finite tiling problem*, in which we are given a Wang tile set that contains a particular blank tile B. The blank tile has all its edges colored identically. Blank tiling of the plane is the trivial tiling where all tiles are blank. A finite tiling is a tiling that contains only a finite number of non-blank tiles. The finite tiling question asks whether a given tile set with the blank tile admits valid finite tilings other than the trivial blank tiling, and this problem is seen undecidable through a simple reduction from the halting problem of Turing machines [13].

Theorem 1.4 *It is undecidable whether a given tile set T with a blank tile admits a valid, finite and non-trivial tiling of the plane.*

Note that the undecidability of the finite tiling problem is much easier to establish than Theorem 1.2. Note also a fundamental difference between the two tiling problems: In the finite tiling problem there is a semi-algorithm to detect if a non-trivial finite tiling exists, while in the general tiling problem it is semi-decidable if a tiling does not exist.

Finally we discuss one particular tile set called SNAKES from [13]. This tile set is aperiodic. The tiles have also an arrow printed on them. The arrow is horizontal or vertical and it points to one of the four neighbors of the tile. Such tiles with arrows are called *directed tiles*. Given any tiling, valid or invalid, the arrows determine paths, obtained by following the arrows printed on the tiles. The tile that follows tile $t(x, y)$ on a path is the neighbor of $t(x, y)$ in the direction indicated by the arrow on $t(x, y)$. Note that a path may enter a loop, or it may visit new tiles indefinitely.

The tile set SNAKES has the following plane filling property: Consider a tiling t and a path P that indefinitely follows the arrows as discussed above. If the tiling is valid at all tiles that P visits, then the path covers arbitrarily large squares. In other words, for every $N \geq 1$ there is a square of $N \times N$ tiles on the plane, all of whose tiles are visited by path P. Note that the tiling may be invalid outside path P, yet the path is forced to snake through larger and larger squares. In fact, SNAKES forces the paths to follow the well-known Peano curve shown in Figure 2.

2 Garden of Eden

One of the earliest discovered properties of cellular automata were the Garden-of-Eden theorems by Moore and Myhill in 1962 and 1963, respectively. These results relate injectivity and surjectivity of CA with each other. A cellular automaton is called *injective* if the global transition function G is one-to-one. It is *surjective* if G is onto. A CA is *bijective* if G is a both onto and one-to-one. We have seen in Corollary 1.1 that bijectivity is equivalent to reversibility.

If G is not surjective then there exist *Garden-of-Eden* configurations, that is, configurations without a pre-image. A trivial property of finite sets is that a function from a set into itself is injective if and only if it is surjective. In cellular automata the same is true only in one-direction: an injective CA is

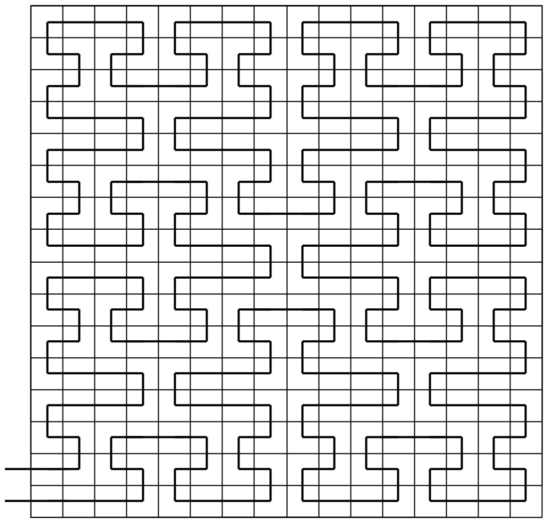


Figure 2: The Peano curve forced by Snakes.

always surjective, but the converse is not true. However, finite configurations behave more like finite sets: G_F is injective if and only if G is surjective. This result, the two directions of which are due to E.F. Moore [18] and J. Myhill [20], is one of the oldest results in the theory of cellular automata:

Theorem 2.1 (Garden-of-Eden theorem [18, 20]) *G_F is injective if and only if G is surjective.*

It is trivial that the injectivity of the full function G implies the injectivity of its restrictions G_F and G_P, so we immediately get the following corollary:

Corollary 2.1 *Injective CA are also surjective. Hence injectivity, bijectivity and reversibility are equivalent.*

It is also easy to see that the surjectivity of G_F or G_P implies the surjectivity of G. This is a direct consequence of the compactness of the configuration space $\mathcal{C}$. The next theorem summarizes these and other known relations. The proofs are straightforward and can be found, for example, in [7]. The results are summarized in Figures 3 and 4.

Theorem 2.2 *The following implications are true in every dimension d:*

- *If G is injective then G_P and G_F are injective,*
- *If G_P or G_F is surjective then G is surjective,*
- *If G_P is injective then G_P is surjective,*

- *If G is injective then G_F is surjective.*

In addition, the following implications are true for one-dimensional CA:

- *If G_P is injective then G is injective,*
- *If G is surjective then G_P is surjective.*

Finally, to establish that some implications are not true we use three cellular automata: one-dimensional automata Xor and Controlled-Xor, and two-dimensional Snake-Xor.

Xor is a one-dimensional radius-$\frac{1}{2}$ cellular automaton with state set $\{0, 1\}$ and the local rule

$$f(x, y) = x + y \pmod 2.$$

State 0 is the quiescent state. Xor is easily seen injective on finite configurations. It is not surjective on finite configurations: for example a configuration with a single state 1 has two infinite predecessors but no finite predecessors.

Controlled-Xor is also a one-dimensional radius-$\frac{1}{2}$ cellular automaton. It has four states 00, 01, 10 and 11. The first bit of each state is a control symbol that does not change. If the control symbol of a cell is 0 then the cell is inactive and does not change its state. If the control symbol is 1 then the cell is active and applies the Xor rule on the second bit. In other words,

$$f(ab, cd) = \begin{cases} ab, & \text{if } a = 0, \\ a(b + d \pmod 2)), & \text{if } a = 1. \end{cases}$$

State 00 is the quiescent state. Controlled-Xor is surjective on finite configurations. It is not injective on unrestricted configurations as two configurations, all of whose cells are active, have the same image if their second bits are complements of each other.

Xor and Controlled-Xor prove the two non-implications in Figure 3. In higher dimensional spaces the rules are applied in one of the dimensions only. Then Xor and Controlled-Xor prove five of the six non-implications in Figure 4. For the remaining

$$G_P \text{ injective } \not\Longrightarrow G \text{ injective}$$

we need Snake-Xor, a two-dimensional cellular automata that uses the Snakes tile set described in Section 1.3.

Snake-Xor is similar to Controlled-Xor. The states consist of two layers: a control layer and a xor layer. The control layer does not change: it only indicates which cells are active and which neighbor cell provides the bit to the Xor operation. In Snake-Xor the control layer consist of Snakes -tiles. Only cells where the tiling on the control layer is valid are active. Active cells execute the modulo two addition on their xor layer. The arrow of the tile tells which neighbor provides the second bit to the Xor operation.

Snake-Xor is not injective: Two configurations c_0 and c_1 whose control layer consist of the same valid tiling have the same image if their xor layers are

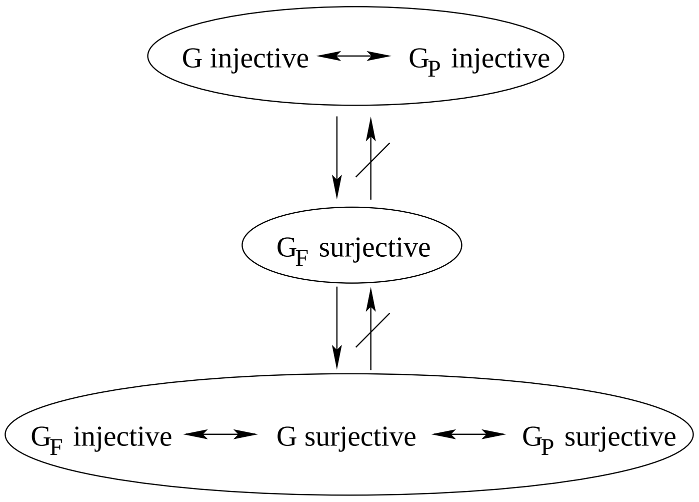


Figure 3: Implications between injectivity and surjectivity properties in one-dimensional CA.

complementary to each other. However, Snake-Xor is injective on periodic configurations, as the plane filling property ensures that on periodic configurations any infinite path that follows the arrows must contain non-active cells.

Notice that Figure 4 contains three implications whose status is unknown.

3 The reversibility and surjectivity questions

Reversibility is a fundamental property of microscopic physical systems, implied by the laws of quantum mechanics. Cellular automata simulating such systems should obey the same laws, hence be reversible. Moreover, a massively parallel computer that optimally uses physics to compute must itself be reversible. Non-reversibility always implies energy dissipation, in practice in the form of heat. It is therefore not surprising that reversible cellular automata have received particular attention since the early days of CA investigation.

Hedlund [8] and Richardson [21] independently proved that all one-to-one cellular automata are reversible (Theorem 1.1). In [24] T. Toffoli demonstrated that any d dimensional cellular automaton can be simulated by a $d+1$ dimensional reversible cellular automaton, which immediately implies the existence of computationally universal, two-dimensional reversible CA. A particularly nice example is the billiard ball CA by N. Margolus [17]. Later it was shown that even reversible one-dimensional CA can be computationally universal [19].

Theorem 3.1 ([19]) *One-dimensional reversible cellular automata exist that are computationally universal.*

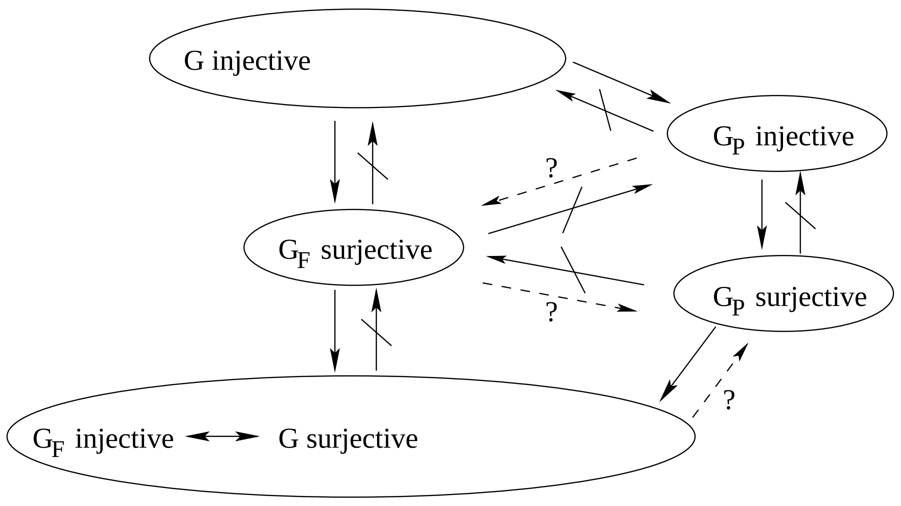


Figure 4: Implications between injectivity and surjectivity properties in two- and higher dimensional CA.

It was determined already in 1972 by S. Amoroso and Y. Patt that it is possible to decide if a given one-dimensional CA is reversible. In the same paper they also provided an algorithm to determine if a given CA is surjective:

Theorem 3.2 ([2]) *There exist algorithms to determine if a given one-dimensional cellular automaton is injective or surjective.*

Elegant decision algorithms based on de Bruijn graphs were later designed by K. Sutner [23]. In higher dimensional spaces the questions are however much harder. It was shown in [10, 13] that

Theorem 3.3 ([13]) *There are no algorithms to determine if a given two-dimensional cellular automaton is injective or surjective.*

The proof for injectivity is a reduction from the tiling problem using the tile set SNAKES from Section 1.3. For a given set T of Wang tiles we construct a two-dimensional CA, similar to SNAKE-XOR of Section 2. The CA has a control layer and a xor layer. The control layes in turn consists of two layers: one with tiles T and one with tiles SNAKES. A cell is active if and only if the tiling is valid at the cell on both tile components. Active cells execute the modulo two addition on their xor layer, and the arrow on the SNAKES tile tells which neighbor provides the second bit to the xor. The plane filling property of SNAKES guarantees that if two different configurations have the same successor then arbitrarily large squares must have a valid tiling. Conversely, if a valid tiling exists then two different configurations with identical control layers can have the same successor. Hence the CA we constructed is injective if and only if T does not admit a valid tiling, and this completes the proof.

The proof concerning the surjectivity is an analogous reduction from the finite tiling problem introduced in Section 1.3. The analogy is based on the fact that surjectivity is equivalent to injectivity on finite configurations (Theorem 2.1). But note the following fundamental difference in the injectivity problems of G and G_F: A semi-algorithm exists for the injectivity of G (based on an exhaustive search for the inverse CA) and for the non-injectivity of G_F (based on looking for two finite configurations with the same image).

Even though Theorem 1.1 guarantees that the inverse function of every injective CA is a cellular automaton, Theorem 3.3 implies that the neighborhood of the inverse CA can be very large: there can be no computable upper bound, as otherwise we could test all candidate inverses one-by-one. In contrast, in the one-dimensional space the inverse automaton can only have a relatively small neighborhood. In one-dimensional radius-$\frac{1}{2}$ cellular automata the inverse neighborhood consists of most $s-1$ consecutive cells where s is the number of states [6], and this bound is tight [11].

4 Limit sets

In cellular automata dynamics there are configurations that are transient in the sense that they can only exist early in the evolution. For example, Garden of Eden configurations can not appear after the first update. The concept of a limit set captures the configurations that are important in the long run, that is, configurations that are not transient.

The limit set $\Lambda = \Lambda[G]$ of cellular automaton G consists of all the configurations that can occur after arbitrarily many computation steps. In other words, it consists of those configurations that are not Garden-of-Eden configurations for any G^n. Define $\Lambda^{(n)} = G^n(\mathcal{C})$ for every $n \geq 1$.Then the limit set of G is

$$\Lambda = \bigcap_{n=1}^{\infty} \Lambda^{(n)}.$$

The finite time sets form a decreasing chain

$$\Lambda^{(1)} \supseteq \Lambda^{(2)} \supseteq \Lambda^{(3)} \supseteq \dots$$

Each $\Lambda^{(n)}$ is compact as an image of the compact set $\mathcal{C}$ under continuous mapping G^n. Consequently, the limit set is compact as an intersection of compact sets.

The limit set can never be empty: it must contain at least one homogeneous configuration. This follows from the fact that every CA has temporally periodic homogeneous configurations. In fact, the limit set is either a singleton set (containing only the quiescent configuration) or it is infinite and contains some non-periodic configurations. It was shown in [4] that if the limit set is a singleton set then $\Lambda = \Lambda^{(n)}$ for some n, that is, all configurations become quiescent in at most n steps, which means that the CA is nilpotent.

It is a natural question to ask what kind of local rules make cellular automata nilpotent. It turns out that no easy characterization exist:

Theorem 4.1 ([4, 12]) *For every $d \geq 1$, it is undecidable whether a given d-dimensional CA is nilpotent or not.*

In two-dimensional case this can be seen as follows [4]: For any given finite set T of Wang tiles we construct a two-dimensional cellular automaton whose state set is $T \cup \{q\}$ and the local update rule keeps the state of a cell unchanged if the tiling is correct at the cell, otherwise the state becomes q. This CA is nilpotent if and only if T does not admit a valid tiling of the plane. The undecidability of the nilpotency problem now follows from the undecidability of the tiling problem (Theorem 1.2).

To show undecidability in the one-dimensional case [12] we use NW-deterministic tiles defined in Section 1.3. For any given NW-deterministic tile set T we construct a one-dimensional CA with state set $T \cup \{q\}$. The neighborhood of the CA is $(0, 1)$ and the local update rule f is defined so that $f(a, b) = c$ if $a, b, c \in T$ and c is a tile that match in color when a and b are placed on its western and northern side, respectively. Since T is NW-deterministic there is at most one matching c for every a and b. In all other cases $f(a, b) = q$. A valid tiling by T is then a valid space-time diagram where the SW/NE diagonals are the configurations. So if a valid tiling exists then the CA is not nilpotent. Conversely, if no valid tiling exist then every starting configuration will produce state q at bounded intervals, and these q's spread to cover the entire line in a finite number of steps. We see that the nilpotency problem must be undecidable as otherwise we could solve the NW-deterministic tiling problem, contradicting Theorem 1.3.

In [4] several properties of the limit sets were proved undecidable. Soon it was discovered that in fact all non-trivial properties of limit sets are undecidable [14]. A property of limit sets simply means any family of cellular automata such that any two cellular automata that have the same limit set (regardless of their state set) either both are in the family or not in the family. The property is non-trivial if the family is not empty but does not contain all cellular automata. In [14] the nilpotency problem was successfully reduced to all non-trivial properties of limit sets, proving that there is no algorithm do determine if the limit set a given CA has the property or not.

Theorem 4.2 (Rice's theorem for limit sets [14]) *For every $d \geq 1$, all non-trivial properties of d-dimensional limit sets are undecidable.*

Note that in the previous theorem the state set of the input CA can be arbitrary. It is an interesting open question to determine what happens when the state set S is fixed. Then surjectivity becomes a property of the limit set: a CA is surjective if and only if its limit set contains all configurations over the state set S. Since surjectivity is decidable in dimension one (Theorem 3.2), there is at least one decidable property of limit sets with restricted alphabet. No other decidable properties are known.

5 Snake tilings

So far we have discussed decision problems that concern various properties of cellular automata. The tile set SNAKES turned out to be a useful tool. In this section we discuss decision problems that concern Wang tiles and that can be proved undecidable using tile set SNAKES .

In the infinite snake tiling problem we are interested to form a valid tiling of infinite chains of consecutive grid positions. More precisely, let $I = \mathbb{Z}$, $I = \mathbb{N}$ or $I = \{1, 2, \dots, n\}$ for some $n \geq 1$. A *snake* is an injective function $s : I \longrightarrow \mathbb{Z}^2$ such that for every $i, i+1 \in I$ positions $s(i)$ and $s(i+1)$ are immediate horizontal or vertical neighbors. Positions $s(i)$ and $s(i+1)$ are called consecutive positions of the snake. Injectivity of s means that the snake may not overlap itself. The snake is called *two-way infinite*, *one-way infinite* or *finite* if $I = \mathbb{Z}$, $I = \mathbb{N}$ or $I = \{1, 2, \dots, n\}$, respectively. In the last case, n is the length of the snake.

Any subset $P \subseteq \mathbb{Z}^2$ of positions is called a *region* of the plane. The region can be finite or infinite. Region P is *connected* if any two positions in P are connected by a snake contained in P. In particular, if $s : I \longrightarrow \mathbb{Z}^2$ is a snake then its range $s(I)$ is a connected region, called a *snake region*.

A tiling of region P is an assignment $f : P \longrightarrow T$ of tiles into all positions in P. Tiling f is valid if the colors of any two neighboring tiles in P match. If $P = s(I)$ is a snake region then a valid tiling of P is called a *strong tiling* of snake s.

Notice that a strong tiling of a snake requires that the tiles in any two neighboring positions of the snake region match, even if the positions are not consecutive in snake s. In other words, if the snake makes a loop and returns back to touch itself then the colors have to match at the touching edges. In contrast, if only the consecutive positions of a snake are required to match then the tiling is a *weak tiling* of the snake. So a weak tiling of snake s is an assignment $f : s(I) \longrightarrow T$ of tiles such that for every $i, i + 1 \in I$ the adjacent edges of tiles in positions $s(i)$ and $s(i + 1)$ have the same color. Notice that a weak tiling of snake s is not necessarily a valid tiling of region $s(I)$, and that a tiling of a snake region P can be either weakly valid or not weakly valid depending on the ordering of the elements of P in the snake.

The following easy observations were proved in [16]:

Theorem 5.1 *Let T be a tile set. The following are equivalent:*

(a) T admits a strong tiling of some two-way infinite snake,

(b) T admits a strong tiling of some one-way infinite snake,

(c) T admits strong tilings of finite snakes of all lengths,

(d) T admits a tiling of some infinite connected region P,

Theorem 5.2 *Let T be a tile set. The following are equivalent:*

(a) T admits a weak tiling of some two-way infinite snake,

(b) T admits a weak tiling of some one-way infinite snake,

(c) T admits weak tilings of finite snakes of all lengths,

In the following we sketch the proof of the following main result from [1]:

Theorem 5.3 *It is undecidable if a given tile set T admits a strong (weak) tiling of an infinite snake.*

Berger's tiling problem will be reduced in two stages. First we prove the undecidability of a directed variant of the problem, which is then reduced further to the actual snake tiling question of Theorem 5.3. Recall that a directed tile is a Wang tile that has an arrow printed on it, pointing to one of the four neighbors. The arrows are used to define snakes: the arrow of a tile specifies which neighbor follows the present tile on the snake.

A strong *directed tiling* of a snake $s : I \longrightarrow \mathbb{Z}^2$ with directed tile set D is an assignment $f : s(I) \longrightarrow D$ of directed tiles to the snake such that the colors match between any two neighboring positions both in $s(I)$, and for every $i, i+1 \in I$ the arrow of tile $f(s(i))$ points to position $s(i+1)$. We only need the strong snake tiling variant here – weak directed tiling could be defined analogously.

Now the plane-filling property of Snakes tiles from Section 1.3 states that any one-way infinite snake s that can be tiled with Snakes necessarily covers arbitrarily large squares. (This statement is slightly stronger than the original plane filling property presented in Section 1.3 because now we do not assume that the neighborhood of the snake can be filled with tiles that match with the tiles on the snake. However, the Snakes tile set in [13] can be modified to satisfy the stronger form of the plane filling requirement.)

Lemma 5.1 *It is undecidable if a given directed tile set admits a strong directed tiling of some one-way infinite snake.*

Proof. We reduce Berger's undecidable tiling problem to the directed snake tiling question: Let T be an arbitrary (undirected) tile set, which is the input instance to the tiling problem. Let us construct directed "sandwich" tiles that consist of two tile components: one tile from T and one from Snakes. In a valid tiling both tile components must match. The direction is inherited from Snakes.

Let us prove that a strong directed tiling of some one-way infinite snake is possible if and only if T admits a tiling. If T correctly tiles the full plane then it also correctly tiles every region, including every snake region. Let us take any snake that Snakes correctly tiles and combine the correct tilings according to T and Snakes to form a valid directed tiling of s.

Conversely, assume that a strong directed tiling of some one-way infinite snake s is possible. Then the plane filling property of Snakes forces s to be plane-filling, that is, s covers arbitrarily large squares. The T components form a valid tiling of $s(I)$ so that T admits valid tilings of squares of all sizes. Therefore T also admits a tiling of the entire plane.

The result now follows from the undecidability of the tiling problem (Theorem 1.2). □

To prove Theorem 5.3 we use a *motif construction.* For any given set D of directed tiles we construct a set T of undirected tiles that admits a tiling (weak or strong) of an infinite snake if and only if D admits a strong directed tiling of an infinite snake. Let n be some large integer. Each directed tile $d \in D$ (called a *macrotile*) will be simulated by a sequence of tiles of T (called *minitiles*) that are forced to form a finite snake (called motif) that follows the borders of an $n \times n$ square (see Figure 5).

On each side of the square the motif forms a bump or a dent. A bump fits inside a dent if they are aligned properly. The south and the east sides of a motif contains a bump and the north and the west sides a dent. The position of the bump and the dent is determined by the color of the corresponding edge in the macrotile. Each color used in D corresponds to a unique location for bumps and dents. As a result, two motifs can be placed side-by-side without overlaps if and only if the colors of the corresponding macrotiles match (see Figure 6).

Each motif has two *free ends* from which it may be linked to another motif. The exit end is in the middle of the side given by the direction of the macrotile, and the entry point is in the middle of any other side of the motif. Hence each macrotile is represented by three different motifs: one for each possible entry direction. The free ends are labeled in such a way that a motif exit may be connected only to the entry of another motif. This can be established by using labels N,E,S and W to indicate direction: label N is used on any exit tile on the north side and any entry tile on the south side of the motifs. The other labels E,S and W are used analogously.

Formation of motifs by minitiles can be forced by using sufficiently many unique colors. Each minitile in every motif is unique, and its colors only fit with the next and the previous minitile in the motif. There is freedom of choice only at the free ends of the motifs where one can choose the next motif to link into the previous motif.

It is easy to see that the minitiles admit a tiling of a one-way infinite snake if and only if D admits a strong directed tiling of a one-way infinite snake. The minitile snake is obtained from the macrotile snake by replacing each macrotile by its motif. Notice that there is no difference between weak and strong tiling of an infinite snake with the minitiles because the tiles are such that no weakly tiled infinite snake can return to touch itself. This completes the sketch of the proof of Theorem 5.3. □

Notice that in the previous construction the minitiles were of the specific type that each tile has two sides that do not match with any other tile. Hence each tile uniquely determines the direction where the snake continues. It follows that the infinite snake tiling question is undecidable even when restricted to such special types of tiles.

In [16] a cyclic variant of the snake tiling problem was introduced. A *cycle* c of length $n \geq 3$ is a snake of length n whose first and last positions are neighbors.

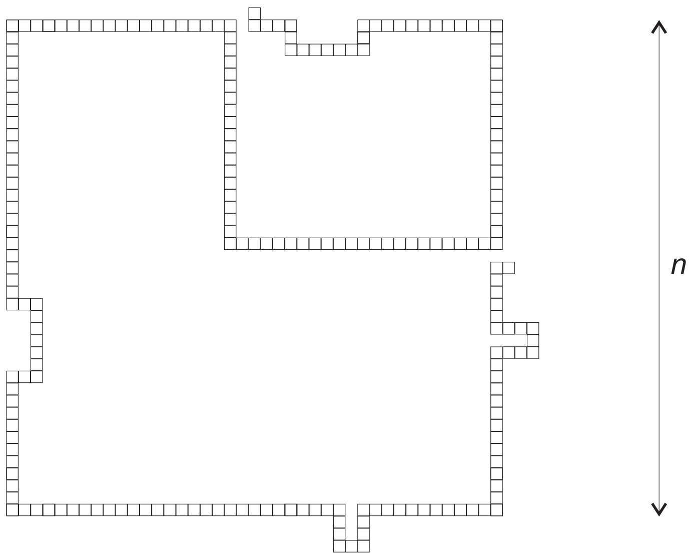


Figure 5: A motif.

In other words, it is an injective function

$$c : I \longrightarrow \mathbb{Z}^2$$

where $I = \{1, 2, \ldots, n\}$, and for all $i \in \mathbb{Z}$ positions $c(i \bmod n)$ and $c(i{+}1 \bmod n)$ are neighbors. A *strong tiling* of cycle c is an assignment

$$f : c(I) \longrightarrow T$$

of tiles in the cycle such that tiles in any two neighboring positions $c(i)$ and $c(j)$ match. A *weak tiling* only requires that tiles in consecutive positions $c(i \bmod n)$ and $c(i + 1 \bmod n)$ match.

We have the following natural decision question: Is there an algorithm to determine if a given tile set admits a valid (weak or strong) tiling of some cycle ? In [16] a technique analogous to the proof of Theorem 5.3 was used to prove the following result:

Theorem 5.4 *It is undecidable if a given tile set T admits a strong (weak) tiling of a cycle.*

In the proof, a reduction from the finite tiling problem of Theorem 1.4 is used instead of Berger's tiling problem. This difference is, once again, reflected also in the following difference in the semi-decidability in infinite and cyclic snake tiling problems: There is a semi-algorithm to determine if some cycle can be tiled, and a semi-algorithm to check if no infinite snake can be tiled. There

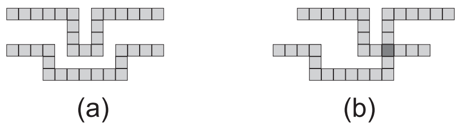


Figure 6: A bump and a dent when the colors (a) match, (b) do not match.

are namely only a countable number of different cycles and they can be easily enumerated. Each cycle can be tiled in a finite number of different ways. In contrast, in the infinite snake tiling problem the negative instances are semi-decidable: if no infinite snake tiling exists then by Theorem 5.1 there exists n such that no finite snake of length n can be tiled. For each n there are only a finite number of different snakes of length n to check.

It would be interesting to find simple, direct reductions between the reversibility question of CA and the infinite snake tiling question, and between the surjectivity question of CA and the cyclic snake tiling question. Existence of such direct reductions seems quite possible: The undecidability proofs were quite similar, they were based on the specific tile set SNAKES and a reduction from the tiling or the finite tiling problem.

Conclusion

We have indicated a close connection between decisions problems concerning Wang tiles and decision problems concerning cellular automata. This is not surprising since both structures are based on a homogeneous rectangular lattice of cells that store a finite element, called either a tile or a state. A major difference, of course, is the fact that cellular automata are dynamic while tilings are static. However, by including the time as a new dimension even the dynamic evolutions of CA can be viewed as tilings of the higher dimensional space. Finally, we have demonstrated the usefulness of the tile set SNAKES, as it was applied in both cellular automata theory and Wang tiles to prove certain properties undecidable.

References

[1] L. Adleman, J. Kari, L. Kari, D. Reishus. On the decidability of self-asssembly of infinite ribbons. Proceedings of FOCS'2002, 43rd Annual Symposium on Foundations of Computer Science, 530–537, 2002

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

[3] R. Berger, The Undecidability of the Domino Problem, Memoirs of the American Mathematical Society 66 (1966).

[4] K. Culik II, J. Pachl and S. Yu, On the limit sets of cellular automata, SIAM Journal on Computing 18 (1989) 831–842.

[5] K. Culik II, An aperiodic set of 13 Wang tiles, Discrete Mathematics 160 (1996) 245–251.

[6] E. Czeizler and J. Kari, A tight linear bound on the neighborhood of inverse cellular automata, to appear.

[7] B. Durand, Global properties of 2D cellular automata, in: Cellular Automata and Complex Systems, E. Goles and S. Martinez (Eds.), Kluwer, 1998.

[8] G. Hedlund, Endomorphisms and automorphisms of shift dynamical systems, Math. Systems Theory 3 (1969) 320–375.

[9] L. P. Hurd, J. Kari and K. Culik, The topological Entropy of Cellular Automata is Uncomputable, Ercodic theory and dynamical systems 12 (1992) 255–265.

[10] J. Kari, Reversibility of 2D cellular automata is undecidable, Physica D 45 (1990) 379–385.

[11] J. Kari, On the Inverse Neighborhoods of Reversible Cellular Automata, in: Lindenmayer Systems, Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology, G. Rozenberg, A. Salomaa (Eds.), 477–495, Springer-Verlag, 1992.

[12] J. Kari, The nilpotency problem of one-dimensional cellular automata. SIAM Journal on Computing 21 (1992) 571–586.

[13] J. Kari, Reversibility and surjectivity problems of cellular automata, Journal of Computer and System Sciences 48 (1994) 149–182.

[14] J. Kari, Rice's theorem for the Limit Sets of Cellular Automata, Theoretical Computer Science 127 (1994) 229–254.

[15] J. Kari, A small aperiodic set of Wang tiles, Discrete Mathematics 160 (1996) 259–264.

[16] J. Kari, Infinite Snake Tiling Problems, In: Proceedings of DLT'2002, Developments in Language Theory, M. Ito, M. Toyama (Eds.), Lecture Notes in computer Science 2450, 67–77, Springer-Verlag, 2003.

[17] N. Margolus, Physics-like models of computation, Physica D 10 (1984) 81–95.

[18] E.F. Moore, Machine Models of Self-reproduction, Proceedings of the Symposium in Applied Mathematics 14 (1962) 17–33.

[19] K. Morita and M. Harao, Computation Universality of one-dimensional reversible (injective) cellular automata, IEICE Transactions E72 (1989) 758–762.

[20] J. Myhill, The Converse to Moore's Garden-of-Eden Theorem, Proceedings of the American Mathematical Society 14 (1963) 685–686.

[21] D. Richardson, Tessellation with local transformations, Journal of Computer and System Sciences 6 (1972) 373–388.

[22] R.M. Robinson, Undecidability and Nonperiodicity for Tilings of the plane, Inventiones Mathematicae 12 (1971) 177–209.

[23] K. Sutner, De Bruijn graphs and linear cellular automata, Complex Systems 5 (1991) 19–31.

[24] T. Toffoli, Computation and construction universality of reversible cellular automata, Journal of Computer and System Sciences 15 (1977) 213–231.

[25] H. Wang, Proving theorems by pattern recognition - II, Bell System Technical Journal 40 (1961) 1–42.

Pattern Formation in Cellular Automata and Array Grammars

Kenichi Morita

Department of Information Engineering, Hiroshima University
Higashi-Hiroshima, 739-8527, Japan
e-mail: morita@iec.hiroshima-u.ac.jp

Abstract

This paper gives a survey on pattern formation problems in cellular automata (CAs) and isometric array grammars (IAGs). Based mainly on our previous works, we discuss how CAs and IAGs can be used for generation of patterns (symbol arrays), and how their abilities are. Among various subclasses of IAGs, we investigate generating abilities of regular array grammars and uniquely parsable array grammars, as well as their relation to CAs. A special type of array generator/recognizer, and self-replication of patterns in reversible CA are also discussed.

1 Introduction

Cellular automata (CAs) and isometric array grammars (IAGs) are useful tools for generating and analyzing multi-dimensional patterns (symbol arrays). Here we discuss how these frameworks can be used for pattern generation, and how their abilities are. An IAG introduced by Rosenfeld [7, 11] is a kind of formal grammar that generates two-dimensional symbol arrays. It can be thought as a variant of tiling system, and also has some similarity to CAs. An isometric regular array grammar (RAG) [1] is a subclass of IAGs having very simple array rewriting rules. In spite of the strong constraint to the form of rewriting rules, RAGs have a rich ability of generating patterns [14]. On the other hand, several decision problems for RAGs become very hard. Especially, the membership problem for RAGs is NP-complete, and thus analysis (parsing) of patterns is intractable [8]. A uniquely parsable isometric array grammar (UPAG) [15] remedies such shortcomings, where any derivation process of a pattern has a "backward deterministic" nature, and hence parsing can be performed deterministically. CAs are also a useful framework for generating and analyzing patterns. We give a kind of pattern generator/recognizer based on reversible partitioned CAs [10]. Finally, we discuss self-reproduction (or self-replication of patterns) in two- and three-dimensional reversible CAs [2, 9].

2 Pattern Formation in Array Grammars

An isometric array grammar (IAG) introduced by Rosenfeld [7, 11] is a formal grammar for two-dimensional languages (see also [13]). An IAG has a set of rules to rewrite symbol arrays. Each rule has an "isometric" property, i.e., both sides of the rule must be the same shape of symbol arrays. This condition is required to avoid a distortion (shear) of an array when applying the rule to a host array. Due to this isometric nature, it acts as a kind of tiling system. It is also regarded as a kind of asynchronous cellular automaton with block-to-block updating rules. In this section, after giving definitions on IAG, we discuss generating abilities of subclasses of IAGs as well as their relation to cellular automata.

2.1 Definitions on Isometric Array Grammars (IAGs)

Let Σ be a finite set of symbols. A *two-dimensional word* over Σ is a non-empty connected array of symbols in Σ. The set of all two-dimensional words over Σ is denoted by Σ^{2+}. Similarly, the sets of all two-dimensional rectangular words and square words are denoted by Σ_R^{2+} and Σ_S^{2+}, respectively.

Definition 2.1 [7, 11] *An* isometric array grammar *(IAG) is defined by the following 5-tuple.*

$$G = (N, T, P, S, \#)$$

N:	*A finite set of nonterminal symbols.*
T:	*A finite set of terminal symbols ($N \cap T = \emptyset$).*
P:	*A finite set of rewriting rules.*
S:	*A start symbol ($S \in N$).*
$\#$:	*A blank symbol ($\# \notin N \cup T$).*

Each rewriting rule in P is of the form $\alpha \to \beta$, and $\alpha, \beta \in (N \cup T \cup \{\#\})^{2+}$ must satisfy the following conditions (to be more precise see [11]):
(1) The shapes of α and β are geometrically identical (i.e., isomeric).
(2) α contains at least one nonterminal symbol.
(3) Terminal symbols in α are not rewritten by the rule $\alpha \to \beta$.
(4) The application of the rule $\alpha \to \beta$ preserves the connectivity of the host array.

A *#-embedded* array of a word $\xi \in (N \cup T)^{2+}$ is an infinite array over $N \cup T \cup \{\#\}$ obtained by embedding ξ in a two-dimensional infinite array of #'s, and is denoted by $\xi_\#$. (Formally, a #-embedded array is a mapping $\mathbf{Z}^2 \to (N \cup T \cup \{\#\})$.) We say that a word η is *directly derived* from a word ξ in G if $\xi_\#$ contains α and $\eta_\#$ is obtained by replacing one of the occurrences of α in $\xi_\#$ with β for some rewriting rule $\alpha \to \beta$ in G. This is denoted by $\xi \underset{G}{\Rightarrow} \eta$. The reflexive and transitive closure of the relation $\underset{G}{\Rightarrow}$ is denoted by $\overset{*}{\underset{G}{\Rightarrow}}$. We say that a word η is *derived* from a word ξ in G if $\xi \overset{*}{\underset{G}{\Rightarrow}} \eta$. The *array language* generated by G is defined by $L(G) = \{ w \mid S \overset{*}{\Rightarrow} w, \text{ and } w \in T^{2+} \}$.

Let $G = (N, T, P, S, \#)$ be an IAG. By restricting the form of a rewriting rule $\alpha \to \beta$ of G, we can obtain three subclasses of IAGs.

Definition 2.2 [7] *If non-# symbols in α are not rewritten into #'s, then G is called a* monotonic array grammar *(MAG).*

Definition 2.3 [1] *If α consists of exactly one nonterminal and possibly some #'s, then G is called a* context-free array grammar *(CFAG).*

Definition 2.4 [1] *If each rewriting rule is one of the following forms, then G is called a* regular array grammar *(RAG), where $A, B \in N$, and $a \in T$.*

$$\# A \;\to\; B\, a, \quad A \# \;\to\; a\, B, \quad \begin{matrix}\# \\ A\end{matrix} \to \begin{matrix}B \\ a\end{matrix}, \quad \begin{matrix}A \\ \#\end{matrix} \to \begin{matrix}a \\ B\end{matrix}, \quad A \;\to\; a.$$

2.2 Pattern Formation in RAGs

It is known that the class of IAGs and its three subclasses form a Chomsky-like hierarchy [1]. The class of RAGs is the smallest one in this hierarchy. However, RAGs have relatively high pattern generating ability in spite of the very restricted form of their rewriting rules. As we shall see later, this generating power comes from the "#-context-sensing ability" of an RAG (i.e., the left-hand side of a rule may have # besides a nonterminal symbol).

Since each rule of an RAG rewrites at most one blank symbol (#) into a non-blank symbol, a large number of rules may be needed to generate a meaningful two-dimensional language. So, it is convenient to introduce a useful subclass of IAGs equivalent to that of RAGs.

Let $r = \alpha \to \beta$ be a rule of a CFAG. r is called *strongly linear* if the following conditions hold (to be more precise, see [14]).
(1) β contains at most one nonterminal.
(2) There is a single-stroke path covering all the symbols of α starting from the position of the nonterminal in α to the (corresponding) position of the nonterminal in β (or to some appropriate position if β has no nonterminal).

Definition 2.5 [14] *Let $G = (N, T, P, S, \#)$ be a CFAG. If every rule in P is strongly linear, then G is called a* strongly linear array grammar *(SLAG).*

Example 2.1 Consider the following rule, where $X, Y \in N$, $a, b \in T$.

$$\begin{matrix} & \# & \# \\ \# & \# & \\ \# & X & \end{matrix} \quad \to \quad \begin{matrix} b & Y \\ a & a \\ b & a \end{matrix}$$

It is easily seen that it is strongly linear. In fact, there is a single-stroke path covering all the six symbols of the left-hand side starting from the X to the upper-right # whose position corresponds to the Y in the right-hand side.

We can decompose the above rule into the following rules of an RAG along the single-stroke path, where X_1, X_2, X_3 and X_4 are new nonterminals.

$$\begin{array}{lcl} \# \, X & \rightarrow & X_1 \, a \\[2ex] X_2 \, \# & \rightarrow & a \, X_3 \\[2ex] X_4 \, \# & \rightarrow & b \, Y \end{array} \qquad \begin{array}{ccc} \begin{array}{c} \# \\ X_1 \end{array} & \rightarrow & \begin{array}{c} X_2 \\ b \end{array} \\[2ex] \begin{array}{c} \# \\ X_3 \end{array} & \rightarrow & \begin{array}{c} X_4 \\ a \end{array} \end{array}$$

It is clear that the above five rules of an RAG correctly simulates the original rule of an SLAG.

Generalizing the method in Example 2.1, the following theorem is obtained. It states that the generating abilities of SLAGs and RAGs are the same (note that the class of RAGs is a subclass of SLAGs).

Theorem 2.1 [14] *For any SLAG G, we can construct an RAG G' such that $L(G) = L(G')$.*

With the aid of SLAGs, we can show that various geometrical patterns such as all rectangles, all squares, etc. are generated by RAGs.

Example 2.2 [14] An SLAG that generates the set of rectangles over $\{a\}$ of size $(6i+4) \times (4j+8)$ $(i, j \in \{0, 1, \cdots\})$.

$$G_R \quad = \quad (\{S, T, L, I, R, B\}, \{a\}, P_R, S, \#)$$

The set P_R consists of the following 12 rules. It is easily verified that all these rules are strongly linear.

(S)

$$\begin{array}{cccc} S & \# & \# & \# \\ \# & \# & & \\ \# & \# & & \end{array} \rightarrow \begin{array}{cccc} a & a & a & T \\ a & a & & \\ a & a & & \end{array}$$

(T1)

$$\begin{array}{ccccc} T & \# & \# & \# & \# \\ & & & \# & \# \\ & & & \# & \# \end{array} \rightarrow \begin{array}{ccccc} a & a & a & a & T \\ & & & a & a \\ & & & a & a \end{array}$$

(T2)

$$\begin{array}{ccccc} T & \# & \# & \# & \# \\ & \# & \# & & \\ & \# & \# & & \end{array} \rightarrow \begin{array}{ccccc} a & a & a & a & R \\ & a & a & & \\ & a & a & & \end{array}$$

(I2)

$$\begin{array}{ccccc} & & \# & \# & \\ & & \# & \# & \\ \# & \# & \# & \# & I \\ \# & \# & & & \\ \# & \# & & & \end{array} \rightarrow \begin{array}{ccccc} & & a & a & \\ & & a & a & \\ a & a & a & a & a \\ a & a & & & \\ L & a & & & \end{array}$$

(I1)

$$\begin{array}{ccccc} \# & \# & & & \\ \# & \# & & & \\ \# & \# & \# & \# & I \\ & & \# & \# & \\ & & \# & \# & \end{array} \rightarrow \begin{array}{ccccc} a & a & & & \\ a & a & & & \\ I & a & a & a & a \\ & & & a & a \\ & & & a & a \end{array}$$

(R1)

$$\begin{array}{cccc} & & & R \\ & & \# & \# \\ & & \# & \# \\ \# & \# & \# & \# \\ \# & \# & & \\ \# & \# & & \end{array} \rightarrow \begin{array}{cccc} & & & a \\ & & a & a \\ & & a & a \\ I & a & a & a \\ a & a & & \\ a & a & & \end{array}$$

(L)

$$\begin{array}{cccc} & & \# & \# \\ L & & \# & \# \\ \# & \# & \# & \# \\ \# & \# & & \\ \# & \# & & \end{array} \rightarrow \begin{array}{cccc} & & a & a \\ & & a & a \\ a & a & a & I \\ a & a & & \\ a & a & & \end{array}$$

(I3)

$$\begin{array}{ccccc} & \# & \# & & \\ & \# & \# & & \\ I & \# & \# & \# & \# \\ & & & \# & \# \\ & & & \# & \# \end{array} \rightarrow \begin{array}{ccccc} & a & a & & \\ & a & a & & \\ a & a & a & a & I \\ & & & a & a \\ & & & a & a \end{array}$$

(I4)

$$\begin{array}{ccccc} & & & \# & \# \\ & & & \# & \# \\ I & \# & \# & \# & \# \\ & \# & \# & & \\ & \# & \# & & \end{array} \rightarrow \begin{array}{ccccc} & & & a & a \\ & & & a & a \\ a & a & a & a & R \\ & a & a & & \\ & a & a & & \end{array}$$

(B2)

$$\begin{array}{ccccc} & & \# & \# & \\ & & \# & \# & \\ \# & \# & \# & \# & B \end{array} \rightarrow \begin{array}{ccccc} & & a & a & \\ & & a & a & \\ a & a & a & a & a \end{array}$$

(B1)

$$\begin{array}{ccccc} \# & \# & & & \\ \# & \# & & & \\ \# & \# & \# & \# & B \end{array} \rightarrow \begin{array}{ccccc} a & a & & & \\ a & a & & & \\ B & a & a & a & a \end{array}$$

(R2)

$$\begin{array}{cccc} & & & R \\ & & \# & \# \\ & & \# & \# \\ \# & \# & \# & \# \end{array} \rightarrow \begin{array}{cccc} & & & a \\ & & a & a \\ & & a & a \\ B & a & a & a \end{array}$$

$$\begin{array}{ccc}
\# & \# & \# \\
\# & S & \# \\
\# & \# & \#
\end{array}
\underset{G_R}{\overset{(S)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & T & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}
\underset{G_R}{\overset{(T1)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & T & \# & \# & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & a & a & \# & \# & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & a & a & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}$$

$$\underset{G_R}{\overset{(T2)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & R & \# \\
\# & a & a & \# & \# & \# & \# & a & a & a & a & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & a & a & a & a & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}
\underset{G_R}{\overset{(R1)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & \# & \# & \# & \# & a & a & a & a & a & a & \# \\
\# & a & a & \# & \# & \# & \# & a & a & a & a & a & a & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & I & a & a & a & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & a & a & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & a & a & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}$$

$$\underset{G_R}{\overset{(I1)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & \# & \# & a & a & a & a & a & a & a & a & \# \\
\# & a & a & \# & \# & a & a & a & a & a & a & a & a & \# \\
\# & \# & \# & \# & \# & I & a & a & a & a & a & a & a & \# \\
\# & \# & \# & \# & \# & \# & \# & a & a & a & a & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & a & a & a & a & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}
\underset{G_R}{\overset{(I2)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & \# & \# & \# & \# & a & a & a & a & \# & \# & \# \\
\# & L & a & \# & \# & \# & \# & a & a & a & a & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}$$

$$\underset{G_R}{\overset{(L)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & \# & \# & a & a & a & a & \# & \# & \# \\
\# & a & a & a & a & \# & \# & a & a & a & a & \# & \# & \# \\
\# & a & a & a & I & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}
\underset{G_R}{\overset{(I3)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & \# & \# & \# \\
\# & a & a & a & a & a & a & a & I & \# & \# & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & a & a & \# & \# & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & a & a & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}$$

$$\underset{G_R}{\overset{(I4)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & R & \# \\
\# & a & a & \# & \# & \# & \# & a & a & a & a & \# & \# & \# \\
\# & a & a & \# & \# & \# & \# & a & a & a & a & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}
\underset{G_R}{\overset{(R2)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & \# & \# & \# & \# & a & a & a & a & a & a & \# \\
\# & a & a & \# & \# & \# & \# & a & a & a & a & a & a & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & B & a & a & a & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}$$

$$\underset{G_R}{\overset{(B1)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & \# & \# & a & a & a & a & a & a & a & a & \# \\
\# & a & a & \# & \# & a & a & a & a & a & a & a & a & \# \\
\# & \# & \# & \# & \# & B & a & a & a & a & a & a & a & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}
\underset{G_R}{\overset{(B2)}{\Rightarrow}}
\begin{array}{cccccccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & a & a & a & a & a & a & a & a & a & a & a & a & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}$$

Figure 1: A derivation process of a rectangular word by G_R of Example 2.2.

Figure 2.2 shows a derivation of a rectangular word of size 10×12.

A derivation process of a rectangular word is as follows. First, the rule (S) is used. Then, (T1) is applied repeatedly (0 times or more) to form the top edge of a rectangle. If (T2) is used, rightward growth of the top edge terminates. At this point "shape codes" are formed on the second and the third rows of the generated array. A shape code consists of a projection and a notch formed by the symbol a's. One bit of information is represented by a pair [projection, notch] or [notch, projection]. The left/right end and the inner part of a word are distinguished by such a pair.

At the right end, either (R1) or (R2) can be used. If (R1) is used, then (I1) is repeatedly applied to grow the inside of a rectangle. It should be noted that the rule (I2) cannot be used in the inside, since the positions of projections and notches do not match between the host array and the left-hand side of the rule. The rule (I2) is used only at the left end to terminate the leftward growth. Note that the shape codes are transmitted to the lower rows after the applications of these rules.

At the left end the rule (L) is used, and then (I3) is repeatedly applied. The rule (I4) is used at the right end to terminate the rightward growth.

If (R2) is applied at the right end, then the downward growth stops. Repeated applications of (B1) makes the bottom edge of a rectangle. The whole derivation process terminates by applying (B2) at the south-west corner of a rectangle. □

By adding appropriates rules to G_R in Example 2.2, we can obtain an SLAG (hence, RAG) that generates $\{a\}_R^{2+}$, the set of all sizes of rectangles. It is also possible to give an SLAG (RAG) that generates the set of all squares.

Theorem 2.2 [14] *There are RAGs that generate $\{a\}_R^{2+}$ and $\{a\}_S^{2+}$.*

2.3 Uniquely Parsable Array Grammars (UPAGs)

As shown in the previous section, RAGs have relatively high generating ability of geometrical patterns. On the other hand, however, several decision problems on RAGs become very hard to solve. It is also due to the #-context-sensing ability. For example, emptiness problem for RAG languages is undecidable [8]. As for the membership problem, the following result is known. Hence, in general, pattern analysis (or parsing) based on RAGs is not performed efficiently.

Theorem 2.3 [8] *The membership problem (given an IAG G and a word $x \in T^{2+}$, decide whether $x \in L(G)$ or not) is NP-complete for the class of RAGs.*

In order to remedy such inefficiency of parsing, a uniquely parsable array grammar (UPAG) was introduced [15]. In this subsection we first give a survey on two-dimensional UPAGs, and then three-dimensional ones.

The subarray of α whose symbols are not changed (*i.e.* rewritten to the same symbols) by the application of $\alpha \rightarrow \beta$ is called the *context portion* of α.

The subarray of α where each symbol is rewritten to a different symbol is called the *rewritten portion* of α. The context portion and the rewritten portion of β are also defined similarly.

Definition 2.6 [15] *Let $G = (N, T, P, S, \#)$ be an IAG. If P satisfies the following conditions, G is called a* uniquely parsable array grammar *(UPAG).*
1. The right-hand side of each rule in P contains a symbol other than $\#$ and S.
2. Let $r_1 = \alpha_1 \to \beta_1$ and $r_2 = \alpha_2 \to \beta_2$ be two rules in P (may be $r_1 = r_2$). Superpose β_1 and β_2 at all the possible positions variously translating them. For any superposition of β_1 and β_2, if all the symbols in overlapping portions of them match, then
(a) these overlapping portions are contained in the context portions of β_1 and β_2, or
(b) the whole β_1 and β_2 are overlapping, and $r_1 = r_2$.

For example, the pair of rules $aB \to ab$ and $Ca \to ca$ satisfies the condition 2(a) of Definition 2.6, but the pair $\#B \to ab$ and $Ca \to ca$ does not.

Example 2.3 [15] A UPAG that generates the set of all squares over $\{a\}$ of size larger than or equal to 2×2.

$$G_S \quad = \quad (\{S, D, G, E\}, \{a\}, P_S, S, \#)$$

The set P_S consists of the following 9 rules.

$$\begin{array}{ll}\# & \# \\ S & \# \\ \# & \end{array} \to \begin{array}{ll}\# & \# \\ a & S \\ S & \end{array} \qquad \begin{array}{lll} & \# & \\ a & S & \# \\ & \# & \end{array} \to \begin{array}{lll} & \# & \\ a & a & \# \\ & G & \end{array} \qquad \begin{array}{lll}\# & S & S \\ & \# & \end{array} \to \begin{array}{lll}\# & a & D \\ & S & \end{array}$$

$$\begin{array}{lll}a & D & S \\ & \# & \end{array} \to \begin{array}{lll}a & a & D \\ & S & \end{array} \qquad \begin{array}{ll}D & G \\ \# & \end{array} \to \begin{array}{ll}a & D \\ G & \end{array} \qquad \begin{array}{ll} & \# \\ D & \# \\ \# & \end{array} \to \begin{array}{ll} & \# \\ a & \# \\ S & \end{array}$$

$$\# S G \to \# a E \qquad a E S \to a a E \qquad \begin{array}{ll}E & \# \\ \# & \end{array} \to \begin{array}{ll}a & \# \\ \# & \end{array}$$

We can verify that G_S is a UPAG (since it is rather tedious to check it, we did it by computer). The following is a derivation example in G_S.

$$S \Rightarrow \begin{array}{ll}a & S \\ S & \end{array} \Rightarrow \begin{array}{lll}a & a & S \\ S & S & \end{array} \Rightarrow \begin{array}{llll}a & a & a & S \\ S & S & S & \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ S & S & S & G \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & D & S & G \\ S & & & \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & D & G \\ S & S & & \end{array}$$

$$\Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & D \\ S & S & G & \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & a \\ S & S & G & S \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & a \\ a & D & G & S \\ S & & & \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & a \\ a & a & D & S \\ S & G & & \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & a \\ a & a & a & D \\ S & G & S & \end{array}$$

$$\Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & a \\ a & a & a & a \\ S & G & S & S \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & a \\ a & a & a & a \\ a & E & S & S \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & a \\ a & a & a & a \\ a & a & E & S \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & a \\ a & a & a & a \\ a & a & a & E \end{array} \Rightarrow \begin{array}{llll}a & a & a & a \\ a & a & a & a \\ a & a & a & a \\ a & a & a & a \end{array}$$

$\square$

A rewriting rule $\alpha \to \beta$ is said to be *reversely applicable* to η *at* (i, j), iff β occurs in $\eta_{\#}$ at the position (i, j), where the position of an occurrence means the *x-y* coordinates of the leftmost symbol of its uppermost row. If $\xi_{\#}$ is obtained by reversely applying $\alpha \to \beta$ at (i, j), we say ξ is *directly reduced* from η by the reverse rewriting with the *label* $L = [\alpha \to \beta,\ (i, j)]$. This is denoted by $\eta \overset{L}{\Leftarrow} \xi$. Apparently, $\eta \overset{L}{\Leftarrow} \xi$ iff $\xi \overset{L}{\Rightarrow} \eta$. If $\eta \overset{L_1}{\Leftarrow} \zeta_1 \overset{L_2}{\Leftarrow} \zeta_2 \cdots \zeta_{n-1} \overset{L_n}{\Leftarrow} \xi$ for some $\zeta_1, \zeta_2, \cdots, \zeta_{n-1}$, we also write it by $\eta \overset{n}{\Leftarrow} \xi$.

The following theorem states that if $\eta \overset{n}{\Leftarrow} S$, then *every* reduction starting from η always reaches the start symbol S in n steps without backtracking.

Theorem 2.4 [15] *Let $G = (N, T, P, S, \#)$ be a UPAG. Let $\alpha \to \beta$ be any rule in P that is reversely applicable to $\eta \in (N \cup T)^{2+}$ at (i, j). If $\eta \overset{n}{\Leftarrow} S$, then for the label $L = [\alpha \to \beta,\ (i, j)]$ there exists a reduction $\ \eta \overset{L}{\Leftarrow} \zeta \overset{n-1}{\Longleftarrow} S\ $ for some ζ.*

This theorem can be generalized to the case of parallel reduction.

Let $G = (N, T, P, S, \#)$ be a UPAG, and let $\alpha_1 \to \beta_1,\ \cdots,\ \alpha_m \to \beta_m$ be rewriting rules in P that are reversely applicable to $\eta \in (N \cup T)^{2+}$ at the positions $(i_1, j_1),\ \cdots,\ (i_m, j_m)$, respectively, and let $L_k = [\alpha_k \to \beta_k,\ (i_k, j_k)]$ $(k = 1, \cdots, m)$. We assume these labels differ each other. Since G is a UPAG, any two of these reverse applications do not overlap except in their context portions. Therefore, these reverse applications can be performed simultaneously (*i.e.* in parallel). We write such parallel reduction by $\eta \overset{\{L_1, \cdots, L_m\}}{\Longleftarrow} \zeta$.

Theorem 2.5 [15] *Let $G = (N, T, P, S, \#)$ be a UPAG. Let $L_1,\ \cdots,\ L_m$ be different labels which are reversely applicable to $\eta \in (N \cup T)^{2+}$. If $\eta \overset{n}{\Leftarrow} S$, then the following reduction exists for some ζ.*

$$\eta \overset{\{L_1, \cdots, L_m\}}{\Longleftarrow} \zeta \overset{n-m}{\Longleftarrow} S$$

It is possible to extend the framework of two-dimensional IAGs to three-dimensional ones. In [2] a three-dimensional UPAG that generates all cubes is given. Figure 2 shows a parallel parsing process of a cube.

2.4 Notes on the Relation to Cellular Automata

There is a close relation between IAGs and CAs. A main difference between IAGs and CAs is that the operation of the former is sequential, while that of the latter is parallel. So, by using a framework of nondeterministic asynchronous CAs, derivation and parsing processes of IAGs can be simulated. Note that, because of the "isometric" nature of IAGs, each rewriting rule of an IAG is directly simulated by a block-to-block updating rule rather than a usual von Neumann neighborhood rule. It is also easy to see that such a CA can be converted into a CA with von Neumann neighborhood by increasing the number of states.

On the other hand, it is possible to use CAs to perform parsing based on UPAGs. By Theorems 2.5, parsing can be derterministically executed by parallel

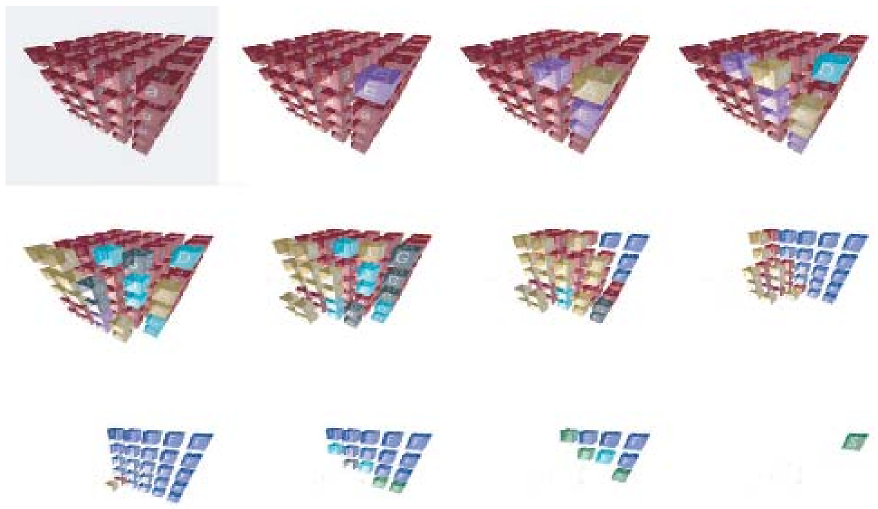


Figure 2: Parallel parsing of a cube based on a three-dimensional UPAG [2].

reverse applications of rewriting rules. Hence, parsing processes of UPAGs can be directly simulated by deterministic synchronous CAs. In fact, Figure 2 can be interpreted as a parallel parsing process performed by a three-dimensional CA.

3 Pattern Formation in Cellular Automata

3.1 Pattern Generation and Parsing by Reversible CAs

Based on the idea noted in Section 2.4, it is possible to give nondeterministic reversible cellular automata that are used both generating and parsing patterns. A *partitioned cellular automata array generator* (PCAAG) [10] is a framework that generates a set of two-dimensional patterns in parallel. If a given PCAAG is reversible, it is also used as a deterministic parsing system like a UPAG.

Figures 3 and 4 shows an example of a PCAAG that generates all rectangles over $\{a\}$ [10]. As shown in Figure 3, a cell is divided into five parts, and the next state of each cell is determined by a center part of this cell, lower part of the upper neighbor cell, left part of the right cell, upper part of the lower cell, and right part of the left cell. This PCAAG is nondeterministic, since two rules have multiple entries of next states. By this, it can generate all sizes of rectangles. In addition it is reversible, since there is no pair of rules that have a common next state on their right-hand side. Hence, its evolution process can be traced backward uniquely, and thus we can use it as a parser (or recognizer) of rectangles.

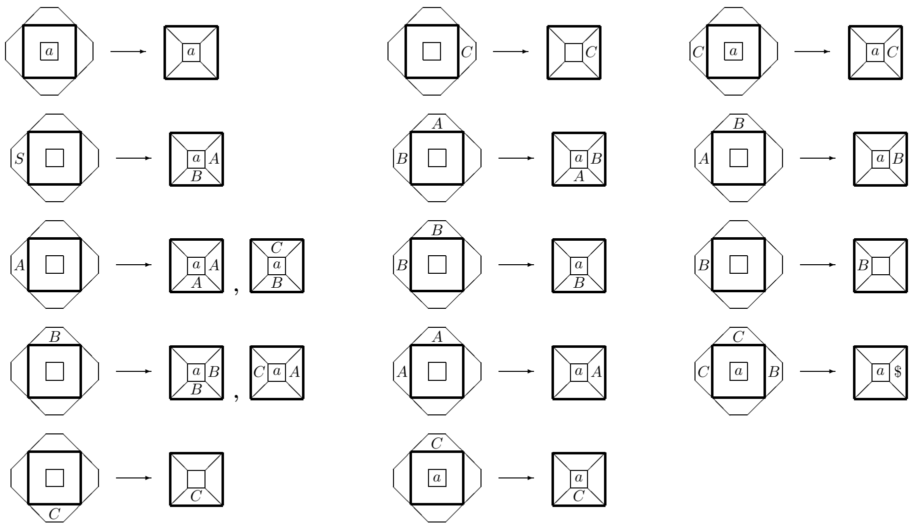


Figure 3: Rules for generating all rectangles by PCAAG [10].

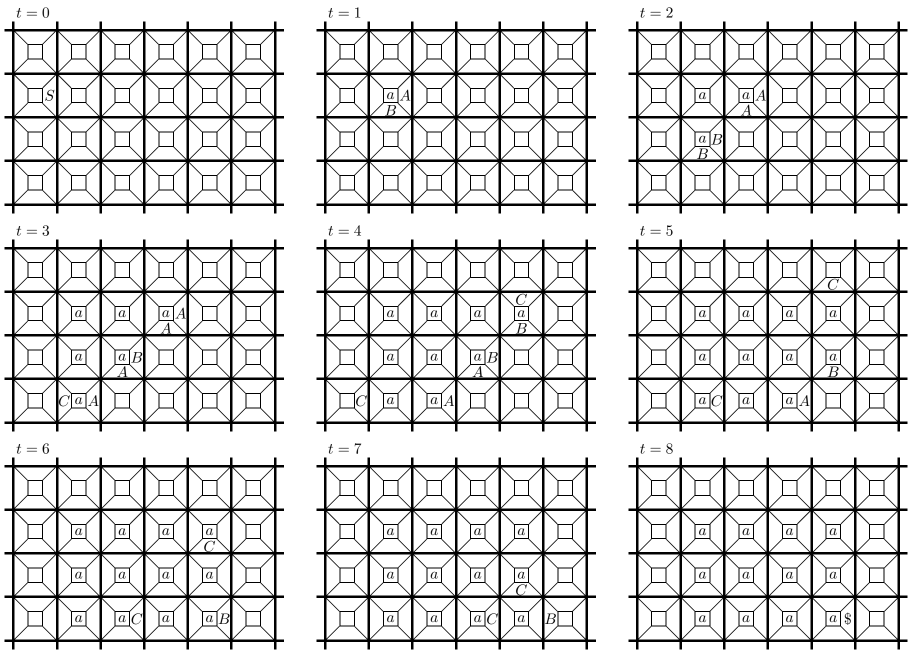


Figure 4: A generating process of a rectangle by PCAAG [10].

3.2 Self-Reproduction in Reversible CA

It is well known that von Neumann [12] first designed a self-reproducing CA. He showed it is possible to construct a Turing machine that can reproduce itself in a 29-state cellular space. Later, Langton [6] showed a very simple self-reproducing cellular automaton by posing so-called Langton's criterion, i.e., self-reproducing objects need not have computation-universality, but the construction of a daughter object should be actively directed by its mother by using its "gene" properly.

Morita and Imai [9] showed that self-reproduction of Langton type is also possible in a reversible CA. They gave a two-dimensional reversible partitioned CA called SR_8. Each cell of SR_8 has 8^5 states (i.e., each of five parts of a cell has 8-state). As in the models of von Neumann and Langton, SR_8 also makes use of a genetic code (description of the object's shape). That is, the body of a daughter object is constructed by interpreting the description.

If a self-reproducing object can encode its shape into a description by checking its body dynamically, there is no need to keep the entire description. In fact, there have been a few models that performs self-reproduction in such a manner [4, 5, 9]. The method employed in SR_8 is called a "shape-encoding" mechanism. By this method, a variety of objects named "Worms" and "Loops" of arbitrary shape can self-reproduce. Hence, SR_8 can be thought as a kind of pattern self-replication mechanism.

A Worm is a simple signal wire with two open ends: a head and a tail. At the tail cell the shape of the Worm is "encoded" into commands and the tail retracts one by one. The commands are sent to the head along the wire. At the head of the Worm, commands are decoded and executed to extend the head. Therefore, it crawls in the space keeping its shape cyclically. By putting a "branch" command, which makes a head branch, a Worm can self-reproduce as in Figure 5.

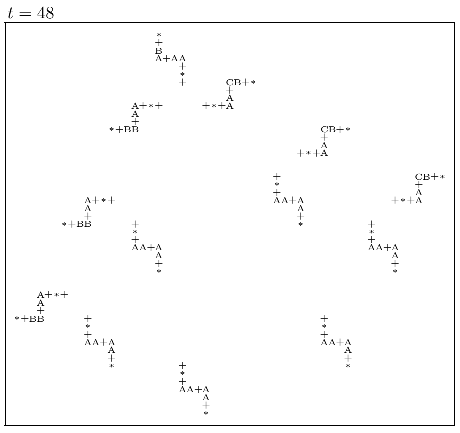


Figure 5: A self-reproducing Worm in SR_8 [9].

A Loop is a simple closed signal wire. It can also reproduce itself in a similar way as in a Worm by extending a "constructing arm" as shown in Figure 6.

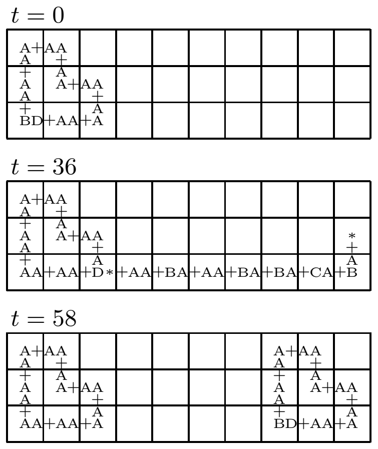


Figure 6: A self-reproducing Loop in SR_8 [9].

It is possible to extend SR_8 to a three-dimensional model. Imai et al. [3] gave a 7-neighbor RPCA called SR_9, and has 9 states in each of seven parts of a cell (hence each cell has 9^7 states). As in SR_8, Worms and Loops of various shapes can reproduce themselves in SR_9.

In the three-dimensional cellular space, varieties of possible shapes and arrangements of Worms and Loops are much greater than that of two-dimensional one. Figure 7 shows a simple self-reproduction process of a three-dimensional Loop. By controlling a position of a constructing arm by a command sequence, we can design a Loop such that it produces a semi-infinite chain as in Figure 8. Another example of a Loop that forms a pile of Loops is shown in Figure 9.

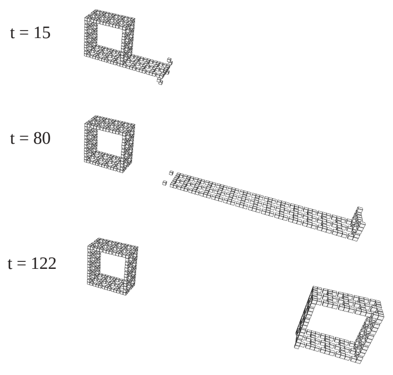


Figure 7: Self-reproduction of a three-dimensional Loop [3].

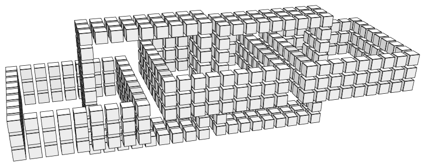


Figure 8: A chain formed by a self-reproducing Loop [3].

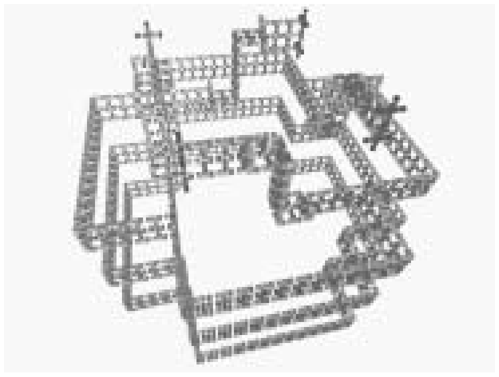


Figure 9: A more complex self-reproducing Loop [3].

4 Concluding Remarks

In this survey we discussed several aspects of pattern formation problems in cellular automata and isometric array grammars. Though these two frameworks have natural similarities, we saw each of them has its own characteristic properties. As for the self-reproducing cellular automata shown here, all reproduced objects are mere patterns, and have no function (e.g., computing) at all. So, it will be interesting to devise a cellular automaton in which functional objects can reproduce themselves in a simple fashion.

Acknowledgement. This work was supported in part by Grant-in-Aid for Scientific Research (C) No. 16500012 from JSPS, and the Telecommunications Advancement Foundation of Japan.

References

[1] Cook, C.R., and Wang, P.S.P, A Chomsky hierarchy of isotonic array grammars and languages, *Computer Graphics and Image Processing*, **8**, 144–152 (1978).

[2] Imai, K., Matsuda, Y., Iwamoto, C., and Morita, K., A three-dimensional uniquely parsable array grammar that generates and parses cubes, *Electronic Notes in Theoretical Computer Science*, **46** (2001).

[3] Imai, K., Hori, T., and Morita, K., Self-reproduction in three-dimensional reversible cellular space, *Artificial Life*, **8**, 155–174 (2002).

[4] 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, 564–576 (1995).

[5] Laing, R., Automaton models of reproduction by self-inspection, *J. Theor. Biol.*, **66**, 437–456 (1977).

[6] Langton, C.G., Self-reproduction in cellular automata, *Physica*, **10D**, 135–144 (1984).

[7] Milgram, D.L., and Rosenfeld, A., Array automata and array grammars, *Information Processing 71*, North-Holland, 69–74 (1972).

[8] Morita, K., Yamamoto, Y., and Sugata, K., The complexity of some decision problems about two-dimensional array grammars, *Information Sciences*, **30**, 241–264 (1983).

[9] Morita, K., and Imai, K., Self-reproduction in a reversible cellular space, *Theoret. Comput. Sci.*, **168**, 337–366 (1996).

[10] Morita, K., Ueno, S, and Imai, K., Characterizing the ability of parallel array generators on reversible partitioned cellular automata, *Int. J. Pattern Recognition and Artificial Intelligence*, **13**, 523–538 (1999).

[11] Rosenfeld, A., *Picture Languages*, Academic Press, New York (1979).

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

[13] Wang, P.S.P. (ed.), *Array Grammars, Patterns and Recognizers*, World Scientific, Singapore (1989).

[14] Yamamoto, Y., Morita, K., and Sugata, K., Context-sensitivity of two-dimensional regular array grammars, *Int. J. Pattern Recognition and Artificial Intelligence*, **3**, 295-319 (1989).

[15] Yamamoto, Y., and Morita, K., Two-dimensional uniquely parsable isometric array grammars, *Int. J. Pattern Recognition and Artificial Intelligence*, **6**, 301-313 (1992).

Contributed Papers

Tilings $\{p, q\}$ of the hyperbolic plane are combinatoric

Kamel Chelghoum[1], Maurice Margenstern[1]*, Benoît Martin[1],
Isabelle Pecci[1], Gencho Skordev[2]

[1] LITA, EA 3097,
UFR MIM, University of Metz,
Île du Saulcy,
57045 Metz, France
e-mail: {chelghou,margens,martin,pecci}@sciences.univ-metz.fr

[2] CeVis, Fachbereich Mathematik,
Universität Bremen,
Universitätsallee, 29, 28359 Bremen, Germany
e-mail: skordev@cevis.uni-bremen.de

November 4, 2004

Abstract

Tilings $\{p, q\}$ of the hyperbolic plane are generated by the reflections in its sides and, recursively, in the sides of its images, of a regular polygon with p sides and with vertex angle $\frac{2\pi}{q}$. We show that these tilings are combinatoric, an important property which was devised by the second author. We investigate also important particular cases when $q = 3$ and $q = 4$ and also when $p = 5$ and $p = 7$.

1 Introduction

1.1

Hyperbolic geometry attracts more and more researchers. Recently, a new combinatorial approach appeared, see [12] and [15] where around thirty papers are quoted on this topic. There are presently much more.

*Corresponding author

Due to the very restricted place of this extended abstract, we cannot insert an introduction to hyperbolic geometry. It is just required that the reader knows what is Poincaré's model in a disc. We refer the reader to [15] for a very short introduction and to [21] for more information.

We first present the *splitting method* and the notion of *combinatoric tiling* which ensues from the former. The method is applied to tilings of the hyperbolic plane or the hyperbolic $3D$ and $4D$ spaces, see [20, 16], which are generated from a single regular tile by reflections in the sides and, recursively, in the sides of the images. The regular tile is a regular polygon in the case of the plane, a regular polytope in higher dimension.

In the second section, we study the classical examples of tilings $\{5,4\}$ and $\{7,3\}$ named, respectively, **rectangular pentagrid** and **ternary heptagrid**, for short, respectively **pentagrid** and **heptagrid**. In a second part of the section, we extend the results to the case of tilings $\{p,4\}$ and $\{p{+}2,3\}$. In the third section, we show how the splitting method applies to the general case of tilings $\{p,q\}$. In the fourth section, we consider the coordinates which are attached to the tiles by the splitting method for the case of tilings $\{p,4\}$ and $\{p{+}2,3\}$. We go then to the general case $\{p,q\}$ in a second part of the section. In the fifth section, we apply these tools to a problem of the localisation of a tile containing a given point and how this can be used for initialising cellular automata in the context of these grids.

1.2 The splitting method

On the basis of [12], a new notion was introduced in [13].

Definition 1.1 *Consider finitely many sets S_0, …, S_k of some geometric metric space X which are supposed to be closed with non-empty interior, unbounded and simply connected. Consider also finitely many closed simply connected bounded set P_1, …, P_h with $h \leq k$. Say that the S_i's and P_ℓ's constitute a* **basis of splitting** *if and only if:*

(i) X splits into finitely many copies of S_0,

(ii) any S_i splits into one copy of some P_ℓ, the **leading tile** *of S_i, and finitely many copies of S_j's,*

where **copy** *means an* **isometric image**, *and where, in the condition (ii), the copies may be of different S_j's, S_i being possibly included.*

As usual, it is assumed that the interiors of the copies of P_ℓ and the copies of the S_j's are pairwise disjoint.

The set S_0 is called the **head** *of the basis and the P_ℓ's are called the* **generating tiles** *and the S_i's are called the* **regions** *of the splitting.*

Consider a basis of splitting of X, if any. We recursively define a tree A which is associated with the basis as follows. First, we split S_0 according to the condition (ii) of the definition 1. This gives us a copy of say P_0 which we call the *root* of A. In the same way, by the condition (ii) of the definition 1, the splitting of each S_i provides us with a copy of some P_ℓ, the leading tile of S_i: these leading tiles are the sons of the root. We say that these tiles and their regions are of the **first generation**. Consider a region S_i of the n^{th} generation. The condition (ii) of the definition provides us with the leading tile P_m of S_i which corresponds to a node of the n^{th} generation, and copies of S_j's which are regions of the $n{+}1^{\text{th}}$ generation. The sons of P_m are the leading tiles of the S_j's being involved in the splitting of S_i. Also, the sons of P_m belong to the $n{+}1^{\text{th}}$ generation.

This recursive process generates an infinite tree with finite branching. This tree, A, is called the **spanning tree of the splitting**, where the *splitting* refers to the basis of splitting $S_0, \ldots, S_k$ with its generating tiles $P_0, \ldots, P_h$.

Definition 2 – *Say that a tiling of X is* **combinatoric** *if it has a basis of splitting and if the spanning tree of the splitting yields exactly the restriction of the tiling to S_0, where S_0 is the head of the basis.*

Now, we shall turn to the application of this method to tilings $\{p, q\}$, starting with particular cases and then studying the general case. The main goal of the paper is to prove the following result:

Theorem 1.1 – *Tilings $\{p, q\}$ are combinatoric.*

From [13], we know that when a tiling is combinatoric, there is a polynomial which is attached to the spanning tree of the splitting.

More precisely, we have the following result:

Theorem 1.2 – (Margenstern, [13]) *Let $\mathcal{T}$ be a combinatoric tiling, and denote a basis of splitting for $\mathcal{T}$ by S_0, …, S_k with $P_0, \ldots, P_h$ as its generating tiles. Let $\mathcal{A}$ be the spanning tree of the splitting. Let M be the square matrix with coefficients m_{ij} such that m_{ij} is the number of copies of S_{j-1} which enter the splitting of S_{i-1} in the condition (ii) of the definition of a basis of splitting. Then the number of nodes of $\mathcal{A}$ of the n^{th} generation are given by the sum of the coefficients of the first row of M^n. More generally, the number of nodes of the n^{th} generation in the tree which is constructed as $\mathcal{A}$ but which is rooted in a node being associated to S_i is the sum of the coefficients of the $i{+}1^{\text{th}}$ row of M^n.*

This matrix, which is an **incidence** matrix of the splitting relation between regions of the basis, is called the **matrix of the splitting**. We call **polynomial of the splitting** the characteristic polynomial of this matrix, being possibly divided by the greatest power of X which it contains as a factor. Denote the polynomial by P. From P, we easily infer the induction equation which allows us to compute very easily the number u_n of nodes of the n^{th} level of $\mathcal{A}$. This also gives us the number of nodes of each kind at this level by the coefficients of M^n on the first row: we use the same equation with different initial values. The sequence $\{u_n\}_{n \in I\!N}$ is called the **recurrent sequence of the splitting**.

First, as in [12, 13], number the nodes of $\mathcal{A}$ level by level, starting from the root and, on each level, from left to right. Second, consider the recurrent sequence of the splitting, $\{u_n\}_{n\geq 1}$: it is generated by the polynomial of the splitting. As we shall see, it turns out that the polynomial has a greatest real root β and that $\beta > 1$. The sequence $\{u_n\}_{n\geq 1}$ is increasing. Now, it is possible to represent any positive number n in the form $n = \sum\limits_{i=0}^{k} a_i.u_i$, where $a_i \in \{0..\lfloor\beta\rfloor\}$, see [2], for instance. The string $a_k \ldots a_0$ is called a representation of n. In general, the representation is not unique and it is made unique by an additional condition: we take the representation which is maximal with respect to the lexicographic order on the words on $\{0..b\}$ where $b = \lfloor\beta\rfloor$. We also call **coordinate** of a tile this representation of the number attached to the tile. The set of coordinates is called the **language of the splitting**.

In section 4, we shall go back to this other side of the general method to complete the theorem.

2 Tilings $\{5, 4\}$, $\{7, 3\}$ and $\{p, 4\}$, $\{p+2, 3\}$

First, we examine the particular cases of tiling $\{5, 4\}$ which we call the **pentagrid**, following [17, 12], and then the properties of tiling $\{7, 3\}$ which we call the **ternary heptagrid**.

2.1 Pentagrid and heptagrid

As was already shown in [17, 12], the pentagrid is combinatoric. Figure 1.a gives a general look of the pentagrid and figure 2.a illustrates the splitting of a quarter of the hyperbolic plane. It shows that the splitting gives rise to two regions: the quarter, S_0 and what we call a **strip**, see region S_1 in figure 2.a.

The spanning tree of the splitting is illustrated in figure 2.a.

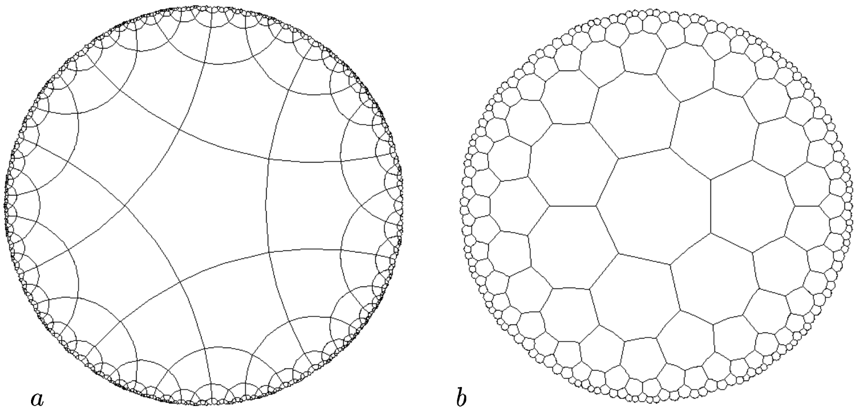


Figure 1: Tilings $\{5, 4\}$, left-hand side, and $\{7, 3\}$, righ-hand side

Looking at figure 2.a, we see that the splitting gives rise to the following incidence matrix:

$$\begin{array}{c c} & \begin{array}{cc} S_0 & S_1 \end{array} \\ \begin{array}{c} S_0 \\ S_1 \end{array} & \left(\begin{array}{cc} 3 & 1 \\ 2 & 1 \end{array}\right) \end{array}$$

The spanning tree of the splitting is generated by simple rules for defining the sons of a node:

1. 3-node or **white** node, in white circle in figure 2.b, at the end of a blue or green arc in figure 2.a, has three sons: to the left, a 2-node, in the middle and to the right, in both cases, 3-nodes.

2. 2-node or **black** node, in black circle in figure 2.b, at the end of a red arc in figure 2.a, has two sons: to the left a 2-node, to the right a 3-node.

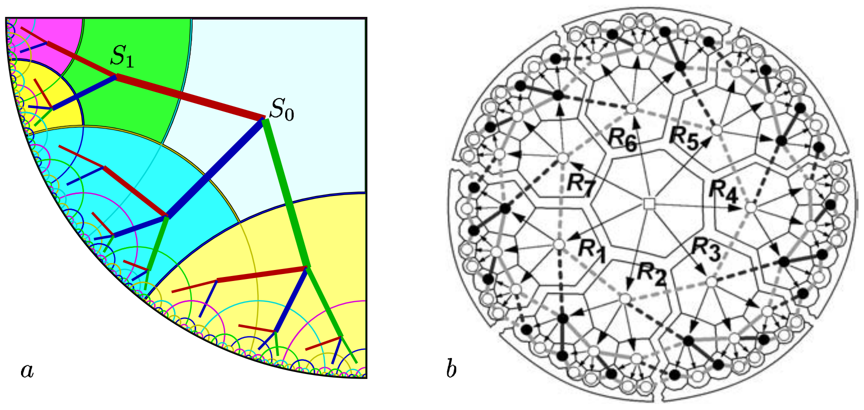


Figure 2: Splitting $\{5, 4\}$, left-hand side, and $\{7, 3\}$, righ-hand side, On the left-hand figure: tiles reached by the tree rooted in S_α, $\alpha \in \{0, 1\}$ constitute region S_α.

From [17, 12], we know that the tree is a spanning tree of the dual graph of the tiling. It is not difficult to restore the dual graph from the tree: we need only to append one edge to each node. The new edge connects a node with the leftmost son of its right-hand neighbour.

On figure 2.b, we see that a very similar splitting holds for tiling $\{7, 3\}$, as it was established in [5]. The proof of this property is illustrated by figure 3, below. It relies on an idea that we shall see in section 3 already used in [18] and on the use of lines of mid-points which pairwise join mid-points of consecutive sides of a heptagon, only two sides per heptagon.

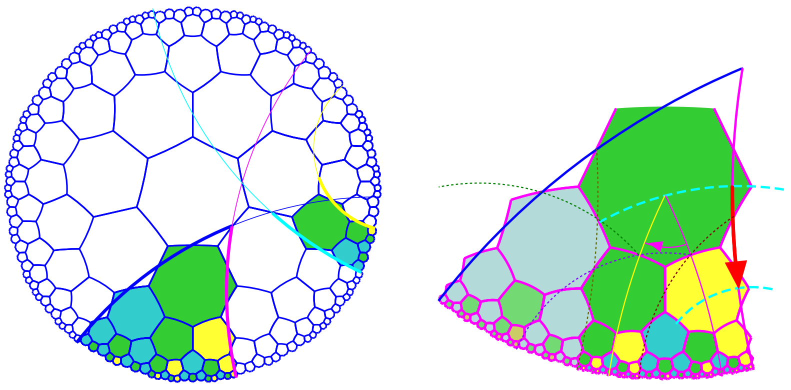


Figure 3: Tiling $\{7, 3\}$: the two regions and mid-point lines; the splitting.

2.2 Rectangular and ternary tilings

We turn now to the case of any number of sides for rectangular regular polygons whose angle vertex is always $\frac{\pi}{2}$. We do also the same for the regular polygons with angle vertex $\frac{2\pi}{3}$ which we call **ternary**.

Indeed, the properties of the pentagrid can be extended to rectangular tilings as was proved in [19]. Namely, we have the same kind of splitting, with a quarter and a strip, and the matrix is:

$$\begin{array}{cc} & \begin{array}{cc} S_0 & S_1 \end{array} \\ \begin{array}{c} S_0 \\ S_1 \end{array} & \left(\begin{array}{cc} p-2 & 1 \\ p-3 & 1 \end{array}\right) \end{array}$$

For tilings $\{p+2, 3\}$, we have a very similar property. The splitting also involves two regions which can be defined using mid-point lines as this was the case for the ternary heptagrid, see figure 4. We get the same matrix for the same value of p when considering the splitting matrix of tiling $\{p, 4\}$.

The reason is that the spanning tree is the same for both tilings. However, the way to restore the dual graph of the tiling from the spanning tree is not the same for both tilings, as this was the case for tilings $\{5, 4\}$ and $\{7, 3\}$. But again this difference is easy to be explained: the basic tile of the dual graph is a quadrangle in the case of tilings $\{p, 4\}$ and it is a triangle in the case of tilings $\{p+2, 3\}$. As a quadrangle is the union of two triangles, we have the connection between the rules applied to the spanning tree to get the dual graph. In the case of tiling $\{p+2, 3\}$, we append a new edge which represents one diagonal of the quadrangle of the dual graph.

This can be seen on figure 5 for the ternary heptagrid. Red dotted edges already come from the case of the pentagrid and they correspond also to a connection which belongs to the dual graph of the heptagrid. Blue dotted edges are new connections induced by the heptagon. They are the above diagonals splitting the quadrangles of the pentagrid dual in two triangles.

3 Numbering and languages of the splitting for tilings $\{p,4\}$ and $\{p+2,3\}$

We remember that when a tiling is combinatoric, we can associate a matrix to the splitting and with the help of the characteristic polynomial of the matrix and its greatest real root, we define the language of the splitting.

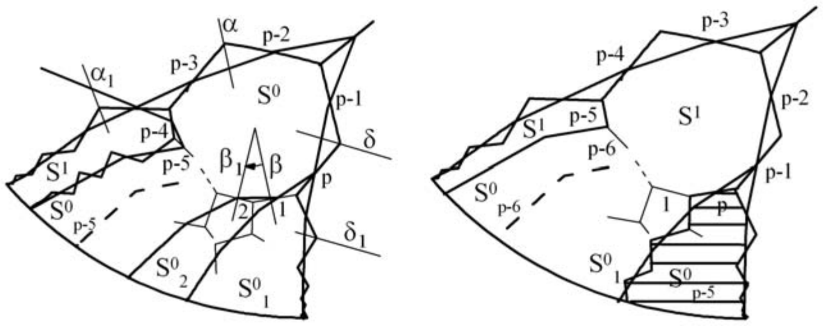


Figure 4: Tiling $\{p+2,3\}$: the two regions and mid-point lines; the splitting.

We use the words of the language as **coordinates** of the tiles by numbering the nodes of the spanning tree from the root to the sons, level by level and, on each level, from left to right. It is enough to associate each tile to the maximal greedy representation of the umber attached to the corresponding node of the spanning tree. In the case of rectangular and ternary regular tilings, we have surprisingly nice properties of this association. We see them first for the pentagrid and the heptagrid and then we generalise to rectangular and to ternary tilings.

3.1 The particular cases: tilings $\{5,4\}$ and $\{7,3\}$

In this case, we do not use exactly the greedy representation associate to the greatest root of the polynomial of the splitting but to the greedy representation with respect to the Fibonacci sequence as the number of tiles at distance n is a constant multiple of the terms of the Fibonacci sequence with odd index. This property was already noticed in [3]. Papers [17] and, mainly [12] shed a new light to explain these properties. What we said in the previous paragraph about the connection between the spanning trees and the dual graphs gives the final reason of the property simply noticed by [3].

The main property of the languages are:

Theorem 3.1 – *The language of the splitting is regular in the case of tilings* $\{5,4\}$ *and* $\{7,3\}$.

The property was proved in [12] for what is the pentagrid. It is a consequence of what we have seen in the previous section and was established in [8].

The result of theorem 3.1 follows from the following property established in [12] and called the **preferred son** property:

Theorem 3.2 *For each node of the spanning tree of the pentagrid, if* ν *is its coordinate, there is exactly one node among its sons whose coordinate is* $\nu 00$ *which is called the* **preferred son**. *Moreover, for white nodes the preferred son is the second son and for black nodes it is the first.*

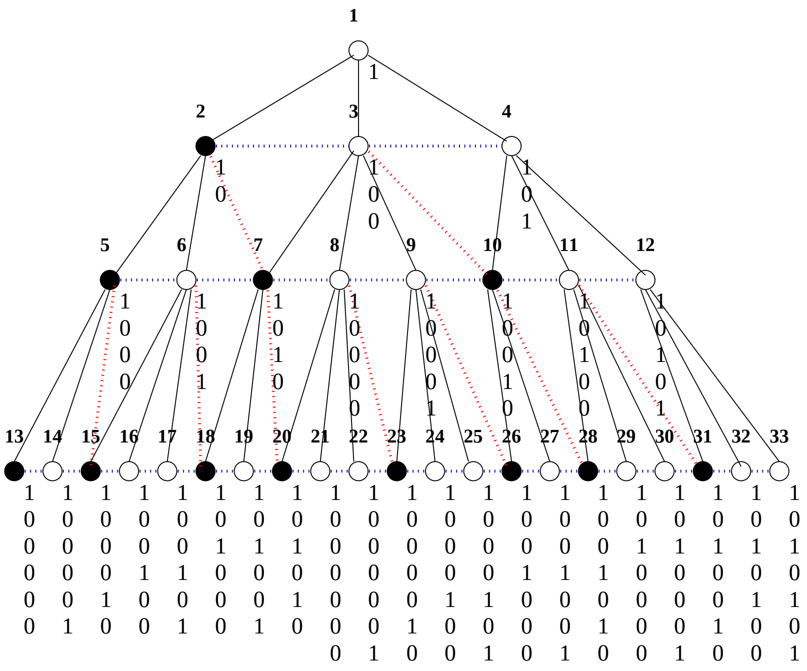


Figure 5: Restoring the dual graph of the ternary heptagrid from its Fibonacci tree

From theorem 3.1, it is possible to devise easy rules to compute the coordinates of the neighbouring tiles of a tile from its coordinate. It is possible also to compute the coordinates of the nodes which are on the path from the node to the root of the spanning tree. Moreover, as this was first indicated in [14], it is possible to perform such computations by linear algorithms. Details of these algorithms are found in [12, 14].

As the preferred son property is a tree property and as it holds for the spanning tree for the pentagrid which is the same as the spanning tree for the ternary heptagrid, see below figure 5, the algorithms of the pentagrid can easily been extended to the case of the ternary heptagrid. A detailed presentation of such an extension is is given in [8, 7].

3.2 The cases of tilings $\{p, 4\}$ and $\{p+2, 3\}$

In the generalised context of tilings $\{p, 4\}$, we consider the maximal representation which is associated to the recurrent relation attached to the greatest real root of the polynomial of the splitting. The property of the preferred son extends to tilings $\{p, 4\}$ as it was done in [19]. Here, if the coordinate of node α is ν, the **continuator** is the node whose coordinate is $\nu 0$. In this context, the preferred son property says that for each node, its continuator appears exactly once among its sons and the corresponding son is said **preferred**.

According to what we reported in section 2, the same property holds for tilings $\{p+2, 3\}$ as the preferred son property is a property of the spanning tree and as the spanning trees are the same for tiling $\{p, 4\}$ and for tiling $\{p+2, 3\}$. The precise formula for the neighbours in the case of tiling $\{p+2, 3\}$ will be given in a forthcoming paper.

4 Tilings$\{p, q\}$

In this section, we shall see that the splitting method can also successfully be applied to the general situation of tilings $\{p, q\}$ for other values of p and q than the one we considered in the previous sections.

However, we have to introduce a distinction between two cases: the case when q is even and the case when it is odd. Notice that this distinction also appears in the study of the groups acting on the hyperbolic plane and whose fundamental domains are regular polygons. Roughly speaking, the situation when both p and q are even is nice, the others are bad, see for instance, [1].

4.1 The splitting when q is even

In this section, we define the splitting of the hyperbolic plane which we use to prove that tiling $\{p, q\}$ is combinatoric when q is even.

We take as S_0, the angular sector of angle $\frac{2\pi}{q}$. Let $q = 2h$. As q is even, we see that the complement of a tile heading the sector can receive $h-1$ copies of S_0 on the left border of the sector, see the left-hand side of figure 6. This can be repeated on the other sides, each time applying a rotation centred in the centre of the tile and of angle $\frac{2\pi}{p}$, until the right-hand border is reached: in that case, what remains is a different region which we call S_1. The right-hand side of figure 6 indicates how S_1 is to be split.

We see that except the borders, everything is similar. On the left-hand border, there only an angle of $(h-2).\frac{2\pi}{q}$ is left instead of $(h-1).\frac{2\pi}{q}$ in the case of S_0. On the right-hand border, a similar reason appears and this prevents us to put S_1 on the same place as in the case of S_0. Hence the splitting indicated by the right-hand side of figure 6.

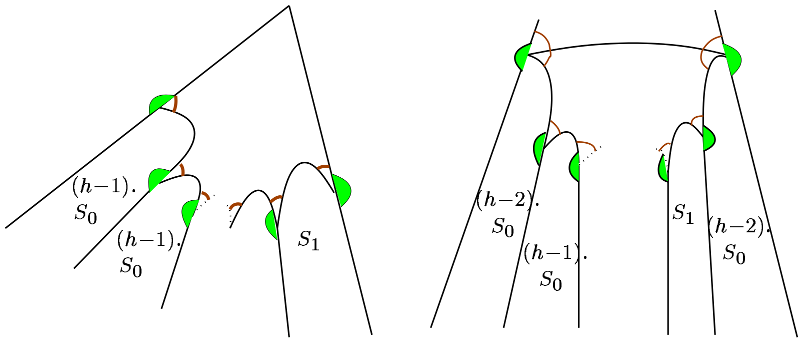


Figure 6: The splitting associated to tiling $\{p, q\}$ when q is even

4.2 The splitting when q is odd

When q is odd, the main problem is that the border defines an angle of π at points A and B, see the left-hand side of figure 7, the case of S_0. Inside S_0, it remains $\pi - \dfrac{2\pi}{q}$ which is not an integral multiple of $\alpha = \dfrac{2\pi}{q}$, the vertex angle of the polygon.

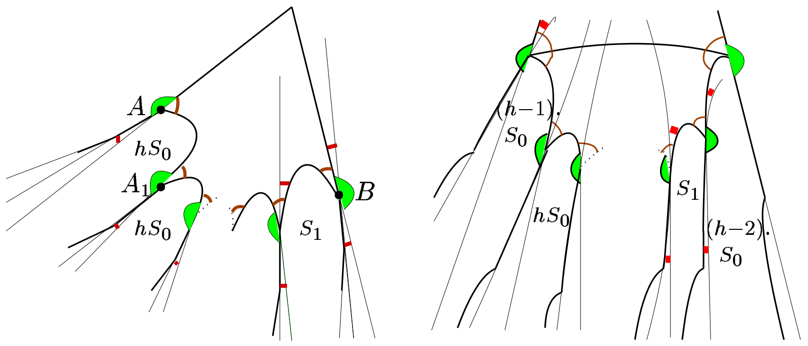


Figure 7: The splitting associated to tiling $\{p, q\}$ when q is odd

There is a way to overcome the difficulty. Let us take the example of vertex A. What is missing at this point is a sector of angle $\dfrac{\pi}{q}$, which is the half of α. If we append such a sector, we get a integral multiple of α. It is not difficult to see that we get then the h copies of S_0 which are announced in figure 7. Now, as we append a half sector, we have to remove it somewhere: we do that on the next vertex, A_1. Doing that, we have in A_1 for the next side the same situation as we had in A: indeed, we obtain it by the already indicated rotation. Finally, we find a region which is similar to S_1, removing also at B a sector of half α.

The same argument works word for word in the case of S_1 which gives the splitting indicated by the right-hand side of figure 7.

4.3 Matrices, polynomials and languages

Using the information provided by figures 6 and 7, we obtain the following matrices of incidence of the splitting, with the even case on the left-hand side matrix and the odd case on the right-hand side one:

$$\begin{array}{c} \\ S_0 \\ S_1 \end{array} \begin{array}{c} \begin{array}{cc} S_0 & S_1 \end{array} \\ \begin{pmatrix} (p-3)(h-1) & 1 \\ (p-2)(h-1)-2 & 1 \end{pmatrix} \end{array} \quad \begin{array}{c} \\ S_0 \\ S_1 \end{array} \begin{array}{c} \begin{array}{cc} S_0 & S_1 \end{array} \\ \begin{pmatrix} (p-3)h & 1 \\ (p-2)h-3 & 1 \end{pmatrix} \end{array}$$

From these matrices, we get the following polynomials:

$$P_0(X) = X^2 - ((p-3)(h-1)+1)X - h + 3, \text{ when } q \text{ is even,}$$
$$P_1(X) = X^2 - ((p-3)h+1)X - h + 3, \text{ when } q \text{ is odd.}$$

And so, we get the following result of [18]:

Theorem 4.1 – (Margenstern, Skordev) *Tilings $\{p,q\}$ are combinatoric and the language of the splitting is regular.*

Consider an initial tile and replicate it by reflection in its sides. By definition, the new tiles belong to the first generation. Next, define generation n+1 as the set of new tiles obtained from the tiles of generation na by reflection in their sides. Notice that in the previous situations, the levels of the spanning tree correspond to the just described notion of generation: nodes of level n correspond to the tiles of generation n which belong to S_0. This is no more the case here, for almost all values of p and q. In this new situation, it is possible to obtain at once nodes which belong to several generation. It is an interesting question to handle the correspondance between levels of the spanning tree and generations in an algorithmic way. This goes out the room available in this extended abstract and it will appear in a forthcoming paper.

5 Localisation and initialisation

From what we established in the previous sections, we have tools to implement cellular automata in any regular grid of the hyperbolic plane, especially in grids associated with tilings $\{p,4\}$ and $\{p+2,3\}$. Reports [10, 11, 6, 7] give a full account of such an implementation in the case of the pentagrid and of the heptagrid. Other interesting implementation details can be found in [4] for the pentagrid and in [14] for the pentagrid again and for a triangular tiling connected with the pentagrid.

First, we look at the localisation problem from which we can solve the initialisation one. We shall see that there is a hidden problem in the localisation: namely a precision problem.

5.1 Localisation

This problem can be stated as follows: *Given a point of the hyperbolic space by its euclidean coordinates in Poincaré's disc, find a tile which contains it.* This problem was first solved in [5] for the pentagrid. In [8], it was also solved for the heptagrid. We have:

Theorem 5.1 – (see [5]) *There is an algorithm which for any point (x, y) in Poincaré's disc with $x^2 + y^2 < 1$ finds a tile of tiling $\{5, 4\}$ in linear time with respect to a binary representation of x, y.*

Indeed, for implemenations, it is clear that points are given as rational numbers in binary representation and not as genuine mathematical real numbers.

The idea of the proof is to consider the successive lines which define the tiling. In the model, these lines are represented by circles whose equation is of the form $X^2 + Y^2 - 2aX - 2bY + 1 = 0$ where (a, b) is the euclidean centre of the circle whose radius is $\sqrt{a^2+b^2-1}$ and we know that $a^2 + b^2 > 1$. As the set of points (u, v) which are outside the circle satisfy $u^2 + v^2 - 2au - 2bv + 1 > 0$, from the splitting process, we easily know in which region of the splitting the point falls. Then, we search such a region that the point falls within its leading pentagon.

However, at this point, we are faced to a precision problem: what is the nature of points (a, b) when the considered circles support lines of the tiling? If we had there *arbitrary* computable real numbers, the problem would turn out to be algorithmically unsolvable: indeed if the point is not on the circle, we shall eventually find whether it is inside or outside the disc. But if the point is on the circle, we may never know due to the undecidability of the equality of computable real numbers with zero.

In [5] a solution was found for the pentagrid which relies on the following remark. The lines defined by sides of tiles in tiling $\{5, 4\}$ are not *any* lines. Going back to the equation of a circle supporting a line in the model, a suitable choice of the first tile allows us to assume that $a, b \in \mathbb{Q}(\vartheta)$, where ϑ is the greatest real number such that $\vartheta^4 + 4\vartheta^2 - 1 = 0$. Now, as it is thoroughly proved in [6], it is not difficult to show that for rationals x, y, it is decidable whether (x, y) belongs to the circle centred at (a, b) when $a, b \in \mathbb{Q}(\vartheta)$. Moreover, as proved in [6], the localisation of (x, y) is linear in the binary representation of x and y.

These arguments were later extended to any tiling $\{p, 4\}$, see [8]. We may also extend them to tilings $\{p+2, 3\}$ at the proce of another remark. Already for tiling $\{7, 3\}$, the application of the previous argument is not straightforward. The reason is that the border of a region is no more defined by a line but by infinitely many ones. However, there is another way which relies on another splitting giving rise to a similar tiling and which is illustrated by figure 8, see also [8, 7].

Theorem 5.2 – (see [8]) *There is an algorithm which for any point (x, y) in Poincaré's disc with $x^2 + y^2 < 1$ finds a tile of tiling $\{7, 3\}$ in linear time with respect to a binary representation of x, y.*

Figure 8 indicates the regions, there are three of them, and it indicates also the elements of the displacements which allow to prove that the regions generate the tiling. Notice that here we have two generating tiles and that the incidence matrix associated to this splitting provides us with the same polynomial of the splitting.

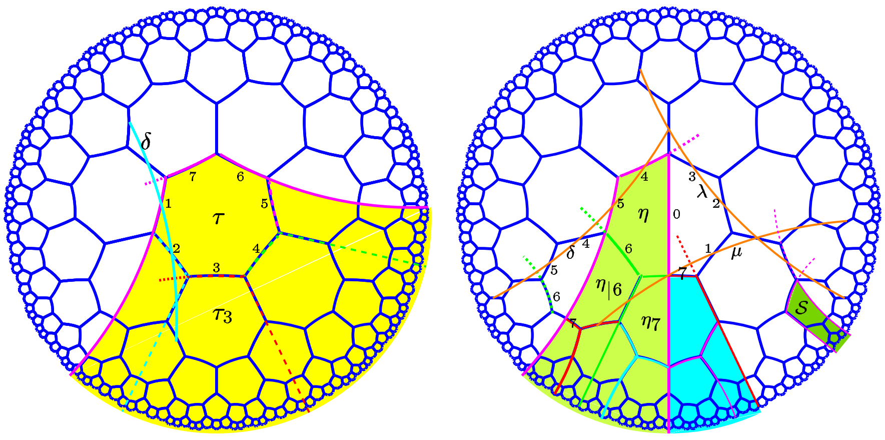


Figure 8: The other splitting associated to tiling $\{7, 3\}$

All these properties can be extended to tilings $\{p, 4\}$ and $\{p{+}2, 3\}$. This was more or less proved in [19] for tilings $\{p, 4\}$. It is also possible to extend them to tilings $\{p{+}2, 3\}$, even for what is the location agorithm. Much more, it is also possible to extend these arguments to the general cases of tilings $\{p, q\}$. This is straightforward for the case of even q's. For the case of odd q's, we can extend the principle of another tiling as suggested by figure 8 which is almost clear for tilings $\{p{+}2, 3\}$. A precise account of all these properties will be given in the announced forthcoming paper.

5.2 Initialisation

In this section, we briefly indicate a general procedure to initialise cellular automata in the regular grids of the hyperbolic plane. The method relies on the localisation algorithm and the following protocole. Assume that the problem which has to be solved by the considered cellular automaton has an initial configuration which is constructed starting from the spatial density of some parameter. A lot of problems solved by cellular automata are of this kind, for instance in modelling diffusion-reaction phenomena. This density can be materialised by the number of particles which can be found in a unit of space. In the case of the hyperbolic plane, a natural frame for this is the number of points which fall in a given tile.

The algorithm of the previous subsection can be used for this purpose. When we know in which tile the point falls, it is enough to update the statistics of that tile. When the set of points is completely performed, it is then easy to compute the denisty.

This situation is more precisely discussed in [5, 8, 9]. It can be also adapted for any tiling $\{p, q\}$ as this was mentioned for the location process. The announced forthcoming paper will also give an account of this point.

Acknowledgement

The corresponding author and the last one thank the University of Metz and the University of Bremen, namely professor Heinz-Otto Peitgen, for giving them the best conditions to their cooperation.

References

[1] H. S. M. Coxeter, W. O. J. Moser, *Generators and Relations for Discrete Groups*, II Ed., Springer, Berlin, (1965).

[2] A.S. Fraenkel, *Systems of numerations*, American Mathematical Monthly, **92** (1985), 105-114.

[3] H. Harborth, *Concentric cycles in mosaic graphs*, Applications of Fibonacci Numbers, **3**, Kluwer Academic Publications, (1990), 123-128.

[4] F. Herrmann, M. Margenstern, *An interactive processing of cellular automata in the hyperbolic plane*, Proceedings of SCI'2002, vol. V, (2002), 406–410.

[5] K. Chelghoum, M. Margenstern, B. Martin, I. Pecci, *Locating Points in the Pentagonal Regular Tiling of the Hyperbolic Plane*, Proceedings of SCI'2003, vol. V, (2003), 25–30.

[6] K. Chelghoum, M. Margenstern, B. Martin, I. Pecci, *Cellular automata in the pentagrid of the hyperbolic plane: tools for interactions*, Publications du LITA, N°2004-101, (2004), 63p.

[7] K. Chelghoum, M. Margenstern, B. Martin, I. Pecci, *Cellular automata in the ternary heptagrid of the hyperbolic plane: tools for interactions*, Publications du LITA, N°2004-101, (2004), 63p.

[8] K. Chelghoum, M. Margenstern, B. Martin, I. Pecci, *Tools for implementing cellular automata in grid* $\{7, 3\}$ *of the hyperbolic plane*, Proceedings of *DMCS*, workshop under *ICALP*'04, Turku, July 2004, (2004), 13–26.

[9] K. Chelghoum, M. Margenstern, B. Martin, I. Pecci, *Cellular automata in the hyperbolic plane: proposal for a new environment*, Lecture Notes in Computer Science, proceedings of ACRI'2004, to appear.

[10] M. Margenstern, *Cellular automata in the hyperbolic plane*, Technical report, Publications du GIFM, I.U.T. of Metz, N°99-103, ISBN 2-9511539-3-7, (1999), 34p.

[11] M. Margenstern, *Cellular automata in the hyperbolic plane, (II)*, Technical report, Publications du GIFM, I.U.T. of Metz, N°2000-101, ISBN 2-9511539-7-X, (2000), 43p.

[12] M. Margenstern, *New Tools for Cellular Automata of the Hyperbolic Plane*, Journal of Universal Computer Science, **6(12)**, (2000), 1226–1252.

[13] M. Margenstern, *A combinatorial approach to infinigons and infinigrids of the hyperbolic plane*, Proceedings of SCI'2002, vol. V, (2002), 417–422.

[14] M. Margenstern, *Implementing Cellular Automata on the Triangular Grids of the Hyperbolic Plane for New Simulation Tools*, Proceedings of the Business and Industry Symposium, ASTC'2003, (2003), 14–21.

[15] M. Margenstern, *Cellular Automata and Combinatoric Tilings in Hyperbolic Spaces. A Survey*, Lecture Notes in Computer Science, **2731**, Proceedings of DMTCS 2003, (2003), 48-72.

[16] M. Margenstern, *The tiling of the hyperbolic* $4D$ *space by the* 120*-cell is combinatoric*, Journal of Universal Computer Science, to appear.

[17] M. Margenstern, K. Morita, *NP problems are tractable in the space of cellular automata in the hyperbolic plane*, Theoretical Computer Science, **259**, (2001), 99-128.

[18] M. Margenstern, G. Skordev, *The tilings* $\{p, q\}$ *of the hyperbolic plane are combinatoric*, Proceedings of SCI'2003, vol. V, (2003), 42–46.

[19] M. Margenstern, G. Skordev, *Fibonacci Type Coding for the Regular Rectangular Tilings of the Hyperbolic Plane*, Journal of Universal Computer Science, **9(5)**, (2003), 398-422.

[20] M. Margenstern, G. Skordev, *Tools for devising cellular automata in the hyperbolic 3D space* Fundamenta Informaticae, **58(2)**, (2003), 369-398.

[21] H. Meschkowski, *Noneuclidean Geometry*, Academic Press, NY, 1964.

Cellular Automata and Parallel Array Systems

Rudolf FREUND* and Fritz TAFILL

Technische Universität Wien
Fakultät für Informatik
Institut für Computersprachen
Favoritenstr. 9, A-1040 Wien, Austria
e-mail: rudi@emcc.at

Abstract

The concept of n-dimensional parallel array systems is a useful means for the formal syntactic description of n-dimensional cellular automata. As n-dimensional parallel array systems are a more general model than n-dimensional cellular automata, they not only allow for the correct formal description of algorithms for n-dimensional cellular automata, but even allow for representing simpler solutions of complex problems.

1 Introduction

As we shall elaborate in this paper, n-dimensional parallel array systems (e.g., see [3]) are useful tools for the formal syntactic description of n-dimensional cellular automata ([15]). For the two-dimensional case, a survey of possible applications of array automata like picture recognition is given in [8]; theoretical results concerning two- and three-dimensional array systems as well as applications of various such models are described in [2] and [14]. As n-dimensional parallel array systems are a more general model than n-dimensional cellular automata, they allow us to construct simpler algorithms for complex problems, e.g., in the area of distributed systems. Attributed parallel array systems have already been shown to be a useful tool for the formal description of the static and dynamic characteristics of neural networks; because of the underlying grid structure, Kohonen's model of self-organizing feature maps (e.g., see [11]) is especially well suited for being represented by n-dimensional parallel array systems (see [5], [6]). Using the concept of different levels of scaling like in [4], even hierarchical networks can be modelled by attributed parallel array systems.

In the second section of this paper we recall the definitions of n-dimensional arrays as well as *n-dimensional parallel array systems (Lindenmayer systems);*

*Corresponding author

moreover, we give a few examples and state some well-known important results. In the third section we show how cellular automata can be represented as parallel array systems. In the fourth section we elaborate various algorithms on cellular automata in the model of n-dimensional parallel array systems as well as attributed n-dimensional parallel array systems. Finally we discuss various extensions of the results presented in this paper.

2 Definitions and Examples

In the main part of this section we introduce the definitions and notations for n-dimensional arrays and parallel array systems; moreover we give some explanatory examples and recall some of the most important results. For basic notions from the theory of formal languages the reader is referred to [13].

2.1 N-dimensional arrays

Let Z denote the set of integers, let N denote the set of positive integers, i.e., $N = \{1, 2, ...\}$, and let $n \in N$. Let V be a (finite) alphabet and let $\#$ be a symbol not in V, which is called the *background* or *blank symbol*. Then an *n-dimensional array* $\mathcal{A}$ over V is a function $\mathcal{A} : Z^n \to V \cup \{\#\}$ with finite support *supp($\mathcal{A}$)*, where $supp(\mathcal{A}) = \{v \in Z^n \mid \mathcal{A}(v) \neq \#\}$. We usually shall write $\mathcal{A} = \{(v, \mathcal{A}(v)) \mid v \in supp(\mathcal{A})\}$.

The set of all n-dimensional arrays over V is denoted by V^{*n}. The *empty array* in V^{*n} with empty support is denoted by Λ_n. Moreover, we define $V^{+n} = V^{*n} - \{\Lambda_n\}$. Any subset of V^{+n} is called a Λ-free n-dimensional array language.

Let $v \in Z^n$. Then the *translation* $\tau_v : Z^n \to Z^n$ is defined by $\tau_v(w) = w + v$ for all $w \in Z^n$, and for any array $\mathcal{A} \in V^{*n}$ we define $\tau_v(\mathcal{A})$, the corresponding n-dimensional array translated by v, by $(\tau_v(\mathcal{A}))(w) = \mathcal{A}(w - v)$ for all $w \in Z^n$. The vector $(0, ..., 0) \in Z^n$ is denoted by Ω_n.

In the literature arrays are often regarded as equivalence classes of arrays with respect to linear translations (e.g., see [10], [12], and [14]), i.e., only the relative positions of the symbols $\neq \#$ in the plane are taken into account, yet in this paper we shall use n-dimensional arrays as defined above.

Example 2.1 *Let* $V = \{a, b\}$, $\mathcal{A} : Z^2 \to \{a, b, \#\}$, $\mathcal{A}(0,1) = \mathcal{A}(1,0) = a$ *and* $\mathcal{A}(0,0) = b$. *Then* $supp(\mathcal{A}) = \{(0,0), (0,1), (1,0)\}$, *and the two-dimensional array* $\mathcal{A}$ *can also be written as* $\mathcal{A} = \{((0,0), b), ((0,1), a), ((1,0), a)\}$.

In order to be able to define the important notion of connectedness of n-dimensional arrays, we need the following definitions:

An (undirected) *graph* g is an ordered pair (K, E) where K is a finite set of nodes and E is a set of undirected edges $\{x, y\}$ with $x, y \in K$. A sequence of different nodes $x_0, x_1, ..., x_m$, $m \in N$, is called a *path* of length m in g with the starting-point x_0 and the ending-point x_m, if for all i with $1 \leq i \leq m$ an edge

$\{x_{i-1}, x_i\}$ in E exists. A graph g is said to be *connected*, if for any two nodes $x, y \in K$, $x \neq y$, a path in g with starting point x and ending point y exists.

Let W be a non-empty finite subset of Z^n. For any $k \in N \cup \{0\}$, a graph $g_k(W) = (W, E_k)$ can be assigned to W such that E_k for $v, w \in W$ contains the edge $\{v, w\}$ if and only if $0 < \|v - w\| \leq k$, where the norm $\|u\|$ of a vector $u \in Z^n$, $u = (u(1), ..., u(n))$, is defined by $\|u\| = \max\{|u(i)| \mid 1 \leq i \leq n\}$. Then W is said to be *k-connected* if $g_k(W)$ is a connected graph. Observe that W is 0-connected if and only if $card(W) = 1$, where $card(W)$ denotes the number of elements in the set W.

Now let V be a finite alphabet and $\mathcal{A}$ an n-dimensional array over V, $\mathcal{A} \neq \Lambda_n$. Then $\mathcal{A}$ is said to be *k-connected* if $g_k(supp(\mathcal{A}))$ is a connected graph; $g_k(supp(\mathcal{A}))$ is called the *k-neighbourhood graph* of $\mathcal{A}$. Obviously, if $\mathcal{A}$ is k-connected then $\mathcal{A}$ is m-connected for all $m > k$, too. The *norm of* $\mathcal{A}$ is the smallest number $k \in N \cup \{0\}$ such that $\mathcal{A}$ is k-connected, and is denoted by $\|\mathcal{A}\|$.

Example 2.2 $\mathcal{A} = \{((0,0), b), ((0,1), a), ((1,0), a) \in \{a,b\}^{+2}$ *is k-connected for every* $k \in N$, *and therefore* $\|\mathcal{A}\| = 1$.

On the other hand, $\mathcal{B} = \{((0,0), a), ((k,k), a)\} \in \{a\}^{+2}$ *is m-connected only for every* $m \geq k$, *and therefore* $\|\mathcal{B}\| = k$.

As many results for n-dimensional arrays for a special n can be taken over immediately for higher dimensions, we introduce the following notion:

Let $n, m \in N$ with $n \leq m$. For $n < m$, the natural embedding $i_{n,m} : Z^n \to Z^m$ is defined by $i_{n,m}(v) = (v, \Omega_{m-n})$ for all $v \in Z^n$; for $n = m$ we define $i_{n,n} : Z^n \to Z^n$ by $i_{n,n}(v) = v$ for all $v \in Z^n$.

To an n-dimensional array $\mathcal{A}$ with $\mathcal{A} = \{(v, \mathcal{A}(v)) \mid v \in supp(\mathcal{A})\}$ we assign the m-dimensional array $i_{n,m}(\mathcal{A}) = \{(i_{n,m}(v), \mathcal{A}(v)) \mid v \in supp(\mathcal{A})\}$.

2.2 N-dimensional parallel array systems (n-dimensional Lindenmayer systems)

An *n-dimensional parallel array production* over an alphabet V is a triple

$$(X, \{(v, X_v) \mid v \in U\}, Y),$$

where U is a finite subset of $Z^n - \{\Omega_n\}$ and X, Y, and X_v, for all $v \in U$, are elements of V. In the sequel, a parallel array production $(X, \{(v, X_v) \mid v \in U\}, Y)$ will also be called an X-production. A finite set T of parallel array productions over the alphabet V is called a *table of parallel array productions*. An array $\mathcal{B}_2 \in V^{*n}$ is said to be *directly derivable* from the array $\mathcal{B}_1 \in V^{*n}$ by T, if and only if for each $w \in Z^n$ a parallel array production $(X, \{(v, X_v) \mid v \in U\}, Y)$ exists such that $\mathcal{B}_1(w) = X$, $\mathcal{B}_2(w) = Y$, and $\mathcal{B}_1(v + w) = X_v$ for all $v \in U$, and we write $\mathcal{B}_1 \Longrightarrow_T \mathcal{B}_2$.

Moreover we shall make the convention that the blank symbol # can only be replaced in the non-empty context by a symbol of V, i.e., $Y \neq \#$ is allowed for $X = \#$ only if $X_v \neq \#$ for some $v \in U$. As a consequence of this convention changes in an n-dimensional array by using a table of parallel productions are restricted to a finite area of the space Z^n.

An *n-dimensional parallel array system* is a quintuple

$$S = (n, V, \Sigma, \#, P),$$

where $n \in N$ is the dimension of the system, $V - \Sigma$ is a set of non-terminal symbols, Σ is a set of terminal symbols, $\#$ is the blank symbol, and P is a finite set of tables of n-dimensional parallel array productions over V.

An array $\mathcal{B}_2 \in V^{*n}$ is said to be *directly derivable* from the array $\mathcal{B}_1 \in V^{*n}$ in S, if and only if there is a table T of n-dimensional parallel array productions in P such that $\mathcal{B}_1 \Longrightarrow_T \mathcal{B}_2$, and we write $\mathcal{B}_1 \Longrightarrow_S \mathcal{B}_2$; $\Longrightarrow_S^*$ denotes the reflexive and transitive closure of the relation $\Longrightarrow_S$.

For a parallel array production of the form $(X, \{(v, X_v) \mid v \in U\}, Y)$, the norm is defined as $\max\{\|v\| \mid v \in U\}$; the *norm of an n-dimensional parallel array system* $S = (n, V, \Sigma, \#, P)$, $\|S\|$, is defined as the maximum of the norms of all productions of tables T in P.

Let $S = (n, V, \Sigma, \#, P)$ be an n-dimensional parallel array system and $A \subset V^{*n}$ be a finite set of *axioms*. Then $G = (S, A)$ is called an *n-dimensional parallel array grammar*. An array $\mathcal{B}_2 \in V^{*n}$ is said to be *directly derivable* from the array $\mathcal{B}_1 \in V^{*n}$ in G, if and only if $\mathcal{B}_1 \Longrightarrow_S \mathcal{B}_2$, and we write $\mathcal{B}_1 \Longrightarrow_G \mathcal{B}_2$; $\Longrightarrow_G^*$ denotes the reflexive and transitive closure of the relation $\Longrightarrow_G$ .The language of n-dimensional arrays generated by G, $L(G)$, is defined by

$$L(G) = \{\mathcal{B} \in \Sigma^{*n} \mid \mathcal{C} \Longrightarrow_G^* \mathcal{B} \text{ for some } \mathcal{C} \in A\}.$$

The norm of G is the maximum of the norms of S and the elements of A.

Example 2.3 *Conway's game of life (e.g., see [7]) can be described by the two-dimensional parallel array system* $S = (2, \{1\}, \{1\}, \#, P)$ *with*

$$\begin{aligned}
P &= P_1 \cup P_2 \cup P_3 \cup P_4, \\
P_1 &= \{(1, \{(v, X_v) \mid v \in U_M\}, 1) \mid card(\{v \in U_M \mid X_v = 1\}) \in \{2,3\}\}, \\
P_2 &= \{(1, \{(v, X_v) \mid v \in U_M\}, \#) \mid card(\{v \in U_M \mid X_v = 1\}) \notin \{2,3\}\}, \\
P_3 &= \{(\#, \{(v, X_v) \mid v \in U_M\}, \#) \mid card(\{v \in U_M \mid X_v = 1\}) \neq 3\}, \\
P_4 &= \{(\#, \{(v, X_v) \mid v \in U_M\}, 1) \mid card(\{v \in U_M \mid X_v = 1\}) = 3\},
\end{aligned}$$

where U_M *is the Moore-neighbourhood* $U_M = \{v \in Z^2 \mid \|v\| = 1\}$; *hence, we obtain* $\|S\| = 1$.

In Figure 1, four successive derivation steps starting from the axiom $\mathcal{A}$ *with*

$$supp(\mathcal{A}) = \{(0,0), (1,0), (2,0), (2,1), (1,2)\}$$

are depicted showing the periodicity of the evolving structures. Thus, for $G = (S, \{\mathcal{A}\})$, $L(G)$ *is infinite, although it contains arrays of only four different structures.*

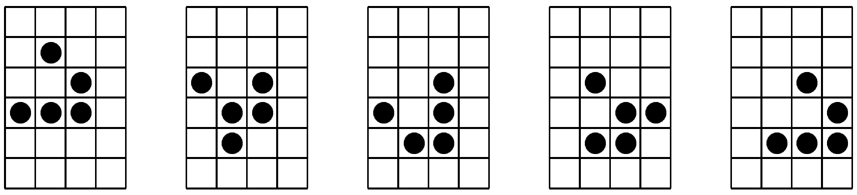


Figure 1: “The Glider”

2.3 N-dimensional attributed parallel array systems

Let V be an alphabet and Γ be a set of *attributes* with $\varepsilon \in \Gamma$, ε being a special attribute called the *empty attribute*. An *n-dimensional attributed array* over (V, Γ) is a pair $(\mathcal{A}, \gamma)$ where $\mathcal{A}$ is an n-dimensional array over V and $\gamma : Z^n \to \Gamma$ is an *attribution function* such that $\gamma(u) = \varepsilon$ for all $u \in Z^n$ with $\mathcal{A}(u) = \#$ (i.e., non-empty attributes can only be assigned to vectors u in the support of $\mathcal{A}$). We shall also write

$$(\mathcal{A}, \gamma) = \{(u, \mathcal{A}(u), \gamma(u)) \mid u \in supp(\mathcal{A})\}.$$

The set of all n-dimensional attributed arrays over (V, Γ) is denoted by $(V, \Gamma)^{*n}$.

An *n-dimensional attributed parallel array production* over (V, Γ) is a pair (p, f), where p is an n-dimensional parallel array production over V, $p = (X, \{(v, X_v) \mid v \in U\}, Y)$, f is a computable function, $f : \Gamma^{U \cup \{\Omega_n\}} \to \Gamma$, U is a finite subset of $Z^n - \{\Omega_n\}$ and X, Y, and X_v, for all $v \in U$, are elements of V. A finite set T of n-dimensional attributed parallel array productions over (V, Γ) is called a *table of n-dimensional attributed parallel array productions.* An attributed array $(\mathcal{B}_2, \gamma_2) \in (V, \Gamma)^{*n}$ is said to be *directly derivable* from the attributed array $(\mathcal{B}_1, \gamma_1) \in (V, \Gamma)^{*n}$ by T, if and only if for each $w \in Z^n$ an attributed parallel array production $((X, \{(v, X_v) \mid v \in U\}, Y), f)$ exists such that $\mathcal{B}_1(w) = X$, $\mathcal{B}_2(w) = Y$, and $\mathcal{B}_1(v + w) = X_v$ for all $v \in U$, as well as $\{(u, \gamma_1(w + u)) \mid u \in U \cup \{\Omega_n\}\}$ is in the domain of f and $\gamma_2(w) = f(\{(u, \gamma_1(w + u)) \mid u \in U \cup \{\Omega_n\}\})$; we write $(\mathcal{B}_1, \gamma_1) \Longrightarrow_T (\mathcal{B}_2, \gamma_2)$.

An *n-dimensional attributed parallel array system* is a sextuple

$$S = (n, V, \Sigma, \#, \Gamma, P),$$

where $n \in N$ is the dimension of the system, $V - \Sigma$ is a set of non-terminal symbols, Σ is a set of terminal symbols, $\#$ is the blank symbol, Γ is a set of attributes, and P is a finite set of tables of n-dimensional attributed parallel array productions over (V, Γ).

An attributed array $(\mathcal{B}_2, \gamma_2) \in (V, \Gamma)^{*n}$ is said to be *directly derivable* from the attributed array $(\mathcal{B}_1, \gamma_1) \in (V, \Gamma)^{*n}$ in S, if and only if there is a table T of n-dimensional attributed parallel array productions in P such that $(\mathcal{B}_1, \gamma_1) \Longrightarrow_T (\mathcal{B}_2, \gamma_2)$, and we write $(\mathcal{B}_1, \gamma_1) \Longrightarrow_S (\mathcal{B}_2, \gamma_2)$; $\Longrightarrow_S^*$ denotes the reflexive and transitive closure of the relation $\Longrightarrow_S$.

Example 2.4 *The absolute positions of the cells in the space Z^2 can be assigned as attributes to the two-dimensional arrays $\mathcal{B}$ we dealt with in Example 2.3 (Conway's game of life) by means of the attribution functions $\gamma_0 : Z^2 \to Z^2 \cup \{\varepsilon\}$ with $\gamma_0(u) = u$ for all $u \in supp(\mathcal{B})$; this information about the absolute positions of the cells in the space Z^2 can be carried over from an axiom by the attributed parallel array productions in the two-dimensional attributed parallel array system $S' = (2, \{1\}, \{1\}, P', \#)$ with $P' = P_1' \cup P_2' \cup P_3' \cup P_4'$, where P_i', $1 \leq i \leq 4$, is obtained from P_i by assigning suitable attribution functions f_i to the parallel array productions in P_i :*

$$\begin{aligned}
P_1' \;=\; & \{((1, \{(v, X_v) \mid v \in U_M\}, 1), f_1) \mid, \\
& \quad card(\{v \in U_M \mid X_v = 1\}) \in \{2, 3\}\}, \\
& f_1(\{(u, \gamma(u)) \mid u \in U_M \cup \{\Omega_2\}\}) = \gamma(\Omega_2), \\
P_2' \;=\; & \{((1, \{(v, X_v) \mid v \in U_M\}, \#), f_2) \mid \\
& \quad card(\{v \in U_M \mid X_v = 1\}) \notin \{2, 3\}\}, \\
& f_2(\{(u, \gamma(u)) \mid u \in U_M \cup \{\Omega_2\}\}) = \varepsilon; \\
P_3' \;=\; & \{((\#, \{(v, X_v) \mid v \in U_M\}, \#), f_3) \mid, \\
& \quad card(\{v \in U_M \mid X_v = 1\}) \neq 3\}, \\
& f_3(\{(u, \gamma(u)) \mid u \in U_M \cup \{\Omega_2\}\}) = \varepsilon; \\
P_4' \;=\; & \{((\#, \{(v, X_v) \mid v \in U_M\}, 1), f_{4,\{(v,X_v) \mid v \in U_M\}}) \mid \\
& \quad card(\{v \in U_M \mid X_v = 1\}) = 3\}, \\
& f_{4,\{(v,X_v) \mid v \in U_M\}}(\{(u, \gamma(u)) \mid u \in U_M \cup \{\Omega_2\}\}) = \gamma(u_0) - u_0 \\
& \text{where } u_0 \text{ is the smallest vector in the set } \{v \in U_M \mid X_v = 1\}, \\
& \text{for arbitrary functions } \gamma : U_M \cup \{\Omega_2\} \to Z^2 \cup \{\varepsilon\}.
\end{aligned}$$

The functions $f_{4,\{(v,X_v) \mid v \in U_M\}}$ guarantee that the correct positions are taken over from cells alive at time t to cells newly born at time $t+1$.

The example given above already shows how models of cellular automata extended by suitable sets of attributes respectively attribution functions can be described by means of attributed parallel array systems. We shall return to such models in the following sections.

3 Representation of Cellular Automata

The example of Conway's game of life (Example 2.3) already indicates how cellular automata can be described by means of n-dimensional parallel array systems. In fact, an n-dimensional *cellular automaton* can be seen as a specific type of an n-dimensional parallel array system meeting special conditions:

An n-dimensional parallel array system $S = (n, V, \Sigma, \#, P)$ is called an *n--dimensional cellular automaton* if and only if it fulfills the following conditions:

1. There exists only one uniform neighbourhood in only one table of parallel array productions in P, i.e., there exists a $U \subseteq Z^n$ such that every parallel array production in P is of the form $(X, \{(v, X_v) \mid v \in U\}, Y)$; in the

following, we shall consider P itself as the single table of parallel array productions.

2. The set of parallel array productions P is deterministic in the sense that if

   $(X, \{(v, X_v) \mid v \in U\}, Y)$ and $(X, \{(v, X'_v) \mid v \in U\}, Y')$

   are two different X-productions with $Y' \neq Y$, then

   $\{(v, X_v) \mid v \in U\} \neq \{(v, X'_v) \mid v \in U\}$.

As derivations in S from a single axiom $\mathcal{C}$ are deterministic, we shall write $S^t(\mathcal{C})$ for the array obtained after t times applying S to $\mathcal{C}$, $t \geq 0$, i.e., recursively we define $S^0(\mathcal{C}) = \mathcal{C}$ and $S^{t+1}(\mathcal{C}) = S(S^t(\mathcal{C}))$, where for any array $\mathcal{B}$ by $S(\mathcal{B})$ we denote the array which is obtained by once applying S to $\mathcal{B}$, i.e., $\mathcal{B} \Longrightarrow_S S(\mathcal{B})$.

An n-dimensional attributed parallel array system $S = (n, V, \Sigma, \#, P)$ with only one table of n-dimensional parallel array productions is called an *n-dimensional (attributed) cellular automaton* if and only if it fulfills the following conditions:

1. There exists only one uniform neighbourhood in the attributed parallel array productions in P, i.e., there exists a $U \subseteq Z^n$ such that every attributed parallel array production in P is of the form $((X, \{(v, X_v) \mid v \in U\}, Y), f)$.

2. The set of attributed parallel array productions P is deterministic in the sense that if

   $((X, \{(v, X_v) \mid v \in U\}, Y), f_1)$ and $((X, \{(v, X'_v) \mid v \in U\}, Y'), f_2)$

   are two different X-productions in P, then either

   (a) $\{(v, X_v) \mid v \in U\} \neq \{(v, X'_v) \mid v \in U\}$ or

   (b) the domains of f_1 and f_2 are disjoint.

Obviously, the two-dimensional (attributed) parallel array system representing Conway's game of life (Example 2.3 and Example 2.4, respectively) is a two-dimensional (attributed) cellular automaton according to this definition.

Usually we only consider n-dimensional parallel array system without extended symbols and without attributes when speaking of cellular automata, yet for some applications this notion is suitable for n-dimensional attributed parallel array systems with extended symbols, too.

4 Algorithms

The formal model of n-dimensional cellular automata introduced in the previous sections can be used to formulate various algorithms in different application fields and to prove these algorithms in this formal setting. In the following we shall give a few characteristic examples for such typical basic algorithms that concern problems to be found in the areas of distributed systems and of cellular automata. For neural networks, especially for Kohonen's model of self-organizing feature maps, specific results were elaborated in [5].

The first result we elaborate can be proved by using n-dimensional cellular automata without attributes, whereas the next algorithms make use of the specific properties offered by the use of suitable sets of attributes. In general, all results hold true for the n-dimensional space Z^n, where n is arbitrary, although in practice $n \in \{1,2,3\}$ might be sufficient for most applications.

In the following lemma we show how we can use n-dimensional cellular automata to check the condition whether all cells in an underlying array are in specific states or not:

Lemma 4.1 *Let $\mathcal{C} \in V^{+n}$ with $\|\mathcal{C}\| = k$, let $w_0 \in supp(\mathcal{C})$ with $\mathcal{C}(w_0) = c$ and $c \neq \mathcal{C}(u)$ for all $u \in supp(\mathcal{C}) - \{w_0\}$, let $V_E \subset V$, and let $t_0 = 2(d+1)$, where d is the maximal distance of a vector $u \in supp(\mathcal{C})$ from w_0 in the k-neighbourhood graph of $\mathcal{C}$. Moreover, let $h^{(i)} : V \to V^{(i)}$, $i \in \{1,2\}$, be morphisms with $h^{(i)}(X) = X^{(i)}$ for all $X \in V$.*

Then we can construct an n-dimensional cellular automaton S with

$$\begin{array}{lcl} S & = & (n, V', \Sigma, \#, P), \\ V' & = & V \cup \left(h^{(1)}(V) \times U_0\right) \cup \left(h^{(2)}(V) \times U_0\right), \\ \Sigma & = & \left(h^{(2)}(V) \times U_0\right), \\ U_0 & = & \{u \in Z^n \mid \|u\| \leq k\}, \end{array}$$

which checks whether $\mathcal{C}(u) \in V_E$ for all $u \in supp(\mathcal{C})$ in such a way that if this condition holds true for $\mathcal{C}$ then

1. *$supp(S^t(\mathcal{C})) = supp(\mathcal{C})$ for all $t \geq 0$;*
2. *$\left(S^{t_0-1}(\mathcal{C})\right)(w_0) \neq \left(S^{t_0}(\mathcal{C})\right)(w_0)$ and $\left(S^{t_0}(\mathcal{C})\right)(w_0) = \left(h^{(2)}(c), \{\Omega_n\}\right)$, which at position w_0 indicates that the condition for $\mathcal{C}$ is fulfilled; moreover, for t_0 we also have*
3. *$S^{t_0+k}(\mathcal{C}) = S^{t_0}(\mathcal{C})$ for all $k > 0$, i.e., $S^{t_0}(\mathcal{C})$ is a stable configuration that is not changed any more by an application of S, and $S^{t_0}(\mathcal{C}) \in \Sigma^{+n}$, i.e., $S^{t_0}(\mathcal{C})$ is a terminal array.*

Proof. Let $U = U_0 - \{\Omega_n\}$; then U is the uniform neighbourhood used in the following parallel array productions in P:

1. $\left(c, \{(u, X_u) \mid u \in U\}, \left(h^{(1)}(c), \Omega_n\right)\right)$
   where $X_u \in V_E \cup \{\#\}$ for every $u \in U$;
2. $\left(X, \{(u, X_u) \mid u \in U\}, \left(h^{(1)}(X), \mu(\{(u, X_u) \mid u \in U\})\right)\right)$
   for all $X \in V$ and all $\{(u, X_u) \mid u \in U\}$ such that
   $P(X, \{(u, X_u) \mid u \in U\})$, and

$(X, \{(u, X_u) \mid u \in U\}, X)$
for all $X \in V$ and all $\{(u, X_u) \mid u \in U\}$ such that
$\neg P(X, \{(u, X_u) \mid u \in U\})$,
where the predicate $P(X, \{(u, X_u) \mid u \in U\})$ is true if and only if
$X \in V_E$ and $X_{u_0} \in \left(h^{(1)}(V) \times U_0\right)$ for some $u_0 \in U$; in this case,
$\mu(\{(u, X_u) \mid u \in U\})$ denotes the smallest vector u in U such that
$X_u \in \left(h^{(1)}(V) \times U_0\right)$;

3. $\left(\left(h^{(1)}(X), u_0\right), \{(u, X_u) \mid u \in U\}, \left(h^{(2)}(X), u_0\right)\right)$
for all $u_0 \in U_0$ and $X \in V$ as well as all
$\{(u, X_u) \mid u \in U\}$ such that $Q(X, \{(u, X_u) \mid u \in U\}, u_0)$, and
$\left(\left(h^{(1)}(X), u_0\right), \{(u, X_u) \mid u \in U\}, \left(h^{(1)}(X), u_0\right)\right)$
for all $u_0 \in U_0$ and $X \in V$ as well as all
$\{(u, X_u) \mid u \in U\}$ such that $\neg Q(X, \{(u, X_u) \mid u \in U\}, u_0)$,
where the predicate $Q(X, \{(u, X_u) \mid u \in U\}, u_0)$ is true if and only if
$X \in V_E$ and for all $u \in U$, $X_u \in \left(\left(h^{(1)}(V_E) \cup h^{(2)}(V_E)\right) \times U_0\right) \cup \{\#\}$
as well as $X_u \in \left(\left(h^{(1)}(V_E) \cup h^{(2)}(V_E)\right) \times \{-u_0\}\right)$ implies
$X_u \in \left(h^{(2)}(V_E) \times \{-u_0\}\right)$;

4. $\left(\left(h^{(2)}(X), u_0\right), \{(u, X_u) \mid u \in U\}, \left(h^{(2)}(X), u_0\right)\right)$
for all $u_0 \in U_0$ and for all $X \in V$, $X_u \in V' \cup \{\#\}$ for all $u \in U$;

5. $(\#, \{(u, X_u) \mid u \in U\}, \#)$ for arbitrary $X_u \in V' \cup \{\#\}$, $u \in U$.

By the production in (1), an impulse from position w_0 is initiated, which is propagated forward from w_0 by the productions in (2) for which the predicate P holds true and which is only possible if the cell under consideration is in a state from V_E. In this way, a tree with root w_0 that is a subgraph of the k-neighbourhood graph of $\mathcal{C}$ is generated, where each non-blank cell remembers its ancestor in the vector $\mu(\{(u, X_u) \mid u \in U\})$. By the productions in (3) this impulse will be reflected from the "borders" of the k-neighbourhood graph; according to the predicate Q, a cell can only take a terminal state from Σ if all its children in this spanning tree generated in the forward propagation period have already changed to a terminal state. The backward propagation successfully ends in w_0 (after exactly $2(d+1)$ steps) if and only if $\mathcal{C}$ fulfills the desired condition. □

Observe that we do not take care of what happens if $\mathcal{C}$ does not fulfill the desired conditions; in this case the derivation with the system S is blocked without yielding a terminal array or a terminal state at w_0. Yet it is an easy exercise to modify the algorithm in such a way that after t_0 steps we reach a stable terminal array which at position w_0 shows by corresponding states whether $\mathcal{C}$ fulfills the desired condition or not.

In a similar way, we can also construct an n-dimensional cellular automaton that allows us to check whether there exists *at least one* non-blank cell in $\mathcal{C}$ which has its state in V_E.

The cellular automaton in the preceding proof was constructed in such a way that the non-blank cells at positions $\neq w_0$ have to wait until the forward impulse from w_0 "activates" them. On the other hand, the algorithm even works if the cells do not work in a synchronized way, although in this case we cannot guarantee the exact upper bound $2(d+1)$.

The following algorithm only works for synchronized cells and uses suitable attributes to guarantee that all cells in a synchronized way take their terminal state after exactly $3d+4$ steps; this problem in an obvious way resembles the FSSP (Firing Squad Synchronization Problem), which is well-known from literature in the area of cellular automata (e.g., see various contributions in [9]). The specific (linear one-dimensional) structure of the squad in the FSSP has allowed the construction of many sophisticated solutions; yet the problem we discuss in the following works for arbitrary dimensions and arrays of arbitrary structures.

Lemma 4.2 *Let $\mathcal{C} \in V^{+n}$ with $\|\mathcal{C}\| = k$, let $w_0 \in supp(\mathcal{C})$ with $\mathcal{C}(w_0) = c$ and $c \neq \mathcal{C}(u)$ for all $u \in supp(\mathcal{C}) - \{w_0\}$, let $V_E \subset V$, and let $t_0 = 3d+4$, where d is the maximal distance of a vector $u \in supp(\mathcal{C})$ from w_0 in the k-neighbourhood graph of $\mathcal{C}$. Then we can construct an n-dimensional attributed cellular automaton $S = (n, V', \Sigma, \#, \Gamma, P)$ which checks whether $\mathcal{C}(u) \in V_E$ for all $u \in supp(\mathcal{C})$ in such a way that if this condition holds true for $\mathcal{C}$ then*

1. *$supp(S^t(\mathcal{C}, \delta)) = supp(\mathcal{C})$ for all $t \geq 0$,*

   where δ is the empty attribution function with $\delta(u) = \varepsilon$ for all $u \in Z^n$;

2. *$S^t(\mathcal{C}, \delta) \in (V' - \Sigma, \Gamma)^{*n}$ for all t with $0 \leq t < t_0$,*

   $S^{t_0}(\mathcal{C}, \delta) \in (\Sigma, \Gamma)^{*n}$, and $S^{t'}(\mathcal{C}, \delta) = S^{t_0}(\mathcal{C}, \delta)$ for all $t' \geq t_0$,

   i.e., in a synchronized way all cells change to a stable terminal state after exactly t_0 steps thus indicating that $\mathcal{C}$ fulfills the desired condition.

Proof. Let $h^{(i)} : V \to V^{(i)}$ for $i \in \{1,2,3,4\}$ denote the morphisms with $h^{(i)}(X) = X^{(i)}$ for all $X \in V$. Then we define

$$U_0 = \{u \in Z^n \mid \|u\| \leq k\}, \ U = U_0 - \{\Omega_n\},$$
$$V' = V \cup \left(\left(\bigcup_{1 \leq i \leq 4} h^{(i)}(V)\right) \times U_0\right),$$
$$\Sigma = \left(h^{(4)}(V_E)\right) \times U_0, \text{ and}$$
$$\Gamma = (N \cup \{0\})^2 \cup \{\varepsilon\}.$$

P contains the following attributed parallel array productions (for arbitrary functions $\gamma : U_0 \to \Gamma$):

1. $\left(\left(c, \{(u, X_u) \mid u \in U\}, \left(h^{(1)}(c), \Omega_n\right)\right), f_1\right)$, where $X_u \in V_E \cup \{\#\}$ for every $u \in U$, and $f_1(\{(u, \gamma(u)) \mid u \in U_0\}) = (0,0)$;

2. $\left(\left(X, \{(u, X_u) \mid u \in U\}, \left(h^{(1)}(X), \mu(\{(u, X_u) \mid u \in U\})\right)\right),\right.$
   $\left. f_{2,\{(u,X_u)|u\in U\}}\right)$ for all $X \in V$ and all $\{(u, X_u) \mid u \in U\}$ such that
   $P(X, \{(u, X_u) \mid u \in U\})$, and
   $((X, \{(u, X_u) \mid u \in U\}, X), f_0)$ for all $X \in V$ and all $\{(u, X_u) \mid u \in U\}$ such that
   $\neg P(X, \{(u, X_u) \mid u \in U\})$,
   where the predicate $P(X\,\{(u, X_u) \mid u \in U\})$ is true if and only if
   $X \in V_E$ and $X_{u_0} \in \left(h^{(1)}(V) \times U_0\right)$ for some $u_0 \in U$; in this case,
   $\mu(\{(u, X_u) \mid u \in U\})$ denotes the smallest vector u in U such that
   $X_u \in \left(h^{(1)}(V) \times U_0\right)$;
   $f_0(\{(u, \gamma(u)) \mid u \in U_0\}) = \gamma(\Omega_n)$;
   $f_{2,\{(u,X_u)|u\in U\}}(\{(u, \gamma(u)) \mid u \in U_0\}) = \gamma(\mu\{(u, X_u) \mid u \in U\}) + (1,0)$;

3. $\left(\left(\left(h^{(1)}(X), u_0\right), \{(u, X_u) \mid u \in U\}, \left(h^{(2)}(X), u_0\right)\right),\right.$
   $\left. f_{3,(\{(u,X(u))|u\in U\},u_0)}\right)$ for all $u_0 \in U_0$ and $X \in V$ as well as all
   $\{(u, X_u) \mid u \in U\}$ such that $Q(X, \{(u, X_u) \mid u \in U\}, u_0)$, and
   $\left(\left(\left(h^{(1)}(X), u_0\right), \{(u, X_u) \mid u \in U\}, \left(h^{(1)}(X), u_0\right)\right), f_0\right)$
   for all $u_0 \in U_0$ and $X \in V$ as well as all
   $\{(u, X_u) \mid u \in U\}$ such that $\neg Q(X, \{(u, X_u) \mid u \in U\}, u_0)$,
   where the predicate $Q(X, \{(u, X_u) \mid u \in U\}, u_0)$ is true if and only if
   $X \in V_E$ and for all $u \in U$, $X_u \in \left(\left(h^{(1)}(V) \cup h^{(2)}(V)\right) \times U_0\right) \cup \{\#\}$
   as well as $X_u \in \left(\left(h^{(1)}(V_E) \cup h^{(2)}(V_E)\right) \times \{-u_0\}\right)$
   implies $X_u \in \left(h^{(2)}(V_E) \times \{-u_0\}\right)$;
   $f_{3,(\{(u,X_u)|u\in U\},u_0)}(\{(u, \gamma(u)) \mid u \in U_0\}) = ((\gamma(\Omega_n))_1,$
   $\max\left(\{(\gamma(\Omega_n))_1\} \cup \left\{(\gamma(u))_2 \mid u \in U, X_u \in \left(h^{(2)}(V) \times \{-u_0\}\right)\right\}\right))$
   where for each $(a_1, a_2) \in (N \cup \{0\})^2$ we define
   $(a_1, a_2)_1 = a_1$ and $(a_1, a_2)_2 = a_2$ as well as $\varepsilon_1 = \varepsilon_2 = \varepsilon$;

4. $\left(\left(\left(h^{(2)}(c), \Omega_n\right), \{(u, X_u) \mid u \in U\}, h^{(3)}(c)\right), f_4\right)$
   for arbitrary $X_u \in \left(h^{(2)}(V_E) \times U_0\right) \cup \{\#\}$, $u \in U$;
   $f_4(\{(u, \gamma(u)) \mid u \in U_0\}) = \gamma(\Omega_n)$;

5. $\left(\left(\left(h^{(2)}(X), u_0\right), \{(u_0, X_{u_0})\} \cup \{(u, X_u) \mid u \in U - \{u_0\}\}, h^{(3)}(X)\right),\right.$
$\left. f_{5,u_0}\right)$ if $X_{u_0} \in \left(h^{(3)}(V_E) \times U_0\right)$ and
$\left(\left(\left(h^{(2)}(c), u_0\right), \{(u_0, X_{u_0})\} \cup \{(u, X_u) \mid u \in U - \{u_0\}\}, h^{(2)}(X)\right), f_0\right)$
if $X_{u_0} \in \left(h^{(2)}(V_E) \times U_0\right)$, for arbitrary $X \in (V_E) - \{c\}$, and
$X_u \in \left(\left(h^{(2)}(V_E) \cup h^{(3)}(V_E)\right) \times U_0\right) \cup \{\#\}$ for $u \in U - \{u_0\}$;
$f_{5,u_0}(\{(u, \gamma(u)) \mid u \in U_0\}) = ((\gamma(\Omega_n))_1, (\gamma(u_0))_2 - 1)$;

6. $\left(\left(\left(h^{(3)}(X), u_0\right), \{(u, X_u) \mid u \in U\}, h^{(3)}(X)\right), f_6\right)$
for arbitrary $X \in V_E$, $u_0 \in U_0$, and
$X_u \in \left(\left(h^{(2)}(V_E) \cup h^{(3)}(V_E)\right) \times U_0\right) \cup \{\#\}$ for $u \in U$;
$$f_6(\{(u, \gamma(u)) \mid u \in U_0\}) = \begin{cases} ((\gamma(\Omega_n))_1, (\gamma(\Omega_n))_2 - 1) \text{ if } \\ \qquad\qquad\qquad\qquad (\gamma(\Omega_n))_2 > 0, \\ \text{else undefined,} \end{cases}$$
i.e., this attributed parallel array production is only applicable if the value of the second component of the attribute of the cell under consideration is positive;

7. $\left(\left(\left(h^{(3)}(X), u_0\right), \{(u, X_u) \mid u \in U\}, h^{(4)}(X)\right), f_7\right)$
for arbitrary $X \in V_E$, $u_0 \in U_0$,
and $X_u \in \left(h^{(3)}(V_E) \times U_0\right) \cup \{\#\}$ for $u \in U$;
$$f_7(\{(u, \gamma(u)) \mid u \in U_0\}) = \begin{cases} ((\gamma(\Omega_n))_1, 0) \text{ if } (\gamma(\Omega_n))_2 = 0, \\ \text{else undefined,} \end{cases}$$
i.e., this attributed parallel array production is only applicable if the value of the second component of the attribute of the cell under consideration is 0;

8. $((\#, \{(u, X_u) \mid u \in U\}, \#), f_\varepsilon)$ for arbitrary $X_u \in V' \cup \{\#\}$, $u \in U$,
$f_\varepsilon(\{(u, \gamma(u)) \mid u \in U_0\}) = \varepsilon$;

9. $\left(\left(\left(h^{(4)}(X), u_0\right), \{(u, X_u) \mid u \in U\}, h^{(4)}(X)\right), f_0\right)$
for arbitrary $X \in V_E$, $u_0 \in U_0$, and $X_u \in \left(h^{(4)}(V_E) \times U_0\right) \cup \{\#\}$, $u \in U$.

The productions in (1), (2) and (3) work in a similar way as already described in the proof of Lemma 4.1, yet now in the first component of the attribute the distances of the cells from w_0 with respect to the k-neighbourhood graph of $\mathcal{C}$ are computed by the productions in (2), whereas in the backward propagation phase the productions in (3) guarantee that after a total number of exactly $2(d+1)$ steps the cell at w_0 knows this maximal distance d, which is propagated in the second components of the attributes. The attributed parallel array production in (4) then starts the "synchronized firing" process, i.e., if d_u is the distance of a cell at position u from w_0 then after exactly d_u further steps by a production in (5) this cell gets to know in the second component of its attribute how long it has to wait before turning into the terminal state from $\left(h^{(4)}\left(V_E\right) \times U_0\right)$. By the productions in (6) this value in the second component is decremented until suitable productions in (7) become applicable, which in a synchronized manner make all cells from $supp(\mathcal{C})$ turn into a terminal state after a total number of exactly $2\,(d+1) + (d+2) = 3d+4$ steps. The productions in (9) guarantee that this terminal array is stable, i.e., we obtain $S^t\left(\mathcal{C},\delta\right) = \left\{\left(u, h^{(4)}\left(\mathcal{C}\left(u\right)\right), \left(d_u, 0\right)\right) \mid u \in supp\left(\mathcal{C}\right)\right\}$ for all $t \geq 3d+4$. □

Whereas the second part of the algorithm described above obviously only works for synchronized cells, the first $2\,(d+1)$ steps again also work for unsynchronized cells, which shows how even in this case the maximal distance in the k-neighbourhood graph of $\mathcal{C}$ from a given cell at position w_0 can be computed and then be realized as an attribute of the cell at position w_0. In order to prove this additional result we chose a more complex algorithm than it would have been necessary to prove the statement of Lemma 4.2 only: As from Lemma 4.1 we already know that it takes exactly $2\,(d+1)$ steps until the impulse initiated from w_0 comes back to w_0, it would be sufficient only to count these steps in the cell at position w_0 and then to propagate this d as shown in the productions in (4) to (9).

The algorithms elaborated in Lemma 4.1 and in Lemma 4.2 were controlled by a specifically marked cell (the so-called "general" in the FSSP), which also allowed us to realize the result of the computation from the state of this single cell only. Yet it is also possible to obtain the results desired in Lemma 4.2, i.e., that in case all cells in the support of $\mathcal{C}$ are in V_E then all cells in the support of $\mathcal{C}$ change to terminal states after a certain number of steps in a synchronized way, without starting from such a specifically marked cell, but instead by an algorithm which starts in parallel in all non-blank cells of the axiom $\mathcal{C}$.

Conclusion

In this paper we have shown how specific variants of the model of n-dimensional (attributed) parallel array systems (e.g., see [3]) can be used for representing n-dimensional cellular automata. Various algorithms for specific problems were elaborated by constructing suitable n-dimensional (attributed) cellular automata and proved in the corresponding formal framework of n-dimensional (attributed) parallel array systems. The ideas of the proofs immediately carry over to other topologies than the Euclidian space Z^n, e.g., to cellular automata on a hexagonal grid (see [1]).

In fact, the algorithms even work on arbitrary graph structures with bounded node degree. A more thorough discussion of these topics is far beyond the scope of this paper, yet offers a wide field for future research. Another possibility for further investigations is to consider special structures of the starting axioms, e.g., n-dimensional full cubes; taking advantage of specific features of such special structures allows for considerable improvements of the general case algorithms elaborated in this paper.

References

[1] K. Aizawa, A, Nakamura, Grammars on the hexagonal array, in: [14], 191–200.

[2] C. H. Chen, L. F. Pau, P. S.-P. Wang (eds.), *Handbook of Pattern Recognition & Computer Vision*, World Scientific Publ., Singapore, 1993.

[3] R. Freund, Aspects of n-dimensional Lindenmayer systems, in: G. Rozenberg, A. Salomaa (eds.), *Developments in Language Theory. At The Crossroads of Mathematics, Computer Science and Biology,* World Scientific Publ., Singapore, 1994, 250–261.

[4] R. Freund, Multi-level eco-array grammars, in: Gh. Păun (ed.), *Artificial Life – Grammatical Models*, Black Sea University Press, Bucureşti, 1995, 166–174.

[5] R. Freund, F. Tafill, Modelling Kohonen networks by attributed parallel array systems, *SPIE'93*, Innsbruck, Austria, 1993.

[6] R. Freund, F. Tafill, Formal representation of neural networks, *ICANN'94*, Sorrento, Italy (1994).

[7] M. Gardner, The fantastic combinations of John Conway's new solitaire game "life", *Mathematical Games, Scientific American* **223**, 1970, 120–123.

[8] K. Inoue, I. Takanami, A survey of two-dimensional automata theory, *Information Sciences* **55**, 1991, 99–121.

[9] M. Kutrib, Th. Worsch (eds.), *Cellular Automata Workshop 1996* (Preproceedings), Schloss Rauischholzhausen, Germany, 1996.

[10] D. L. Milgram, A. Rosenfeld, Array automata and array grammars, *Inform. Processing '71*, North-Holland, 1972, 69–74.

[11] H. Ritter, K. Schulten, T. Martinetz, *Neuronale Netze*, Addison-Wesley, Deutschland, 1990.

[12] A. Rosenfeld, *Picture Languages*, Academic Press, Reading, MA, 1979.

[13] A. Salomaa, *Formal Languages*, Academic Press, Reading, MA, 1973.

[14] P. S.-P. Wang (ed.), *Array Grammars, Patterns and Recognizers*, World Scientific Series in Computer Science **18**, World Scientific Publ., Singapore, 1989.

[15] S. Wolfram (ed.), *Theory and applications of cellular automata*, World Scientific Publ., Singapore, 1986.

The geometry of Penrose tilings: projection and renormalization

Jeroen S. Lamb, Edmund Harriss

Imperial College, London
London SW7 2AZ
e-mail: jeroen.lamb@imperial.ac.uk

October 14, 2004

Abstract

In the early 1970's, R. Penrose constructed a set of two tiles that can tile the plane only nonperiodically. His proof uses a renormalization argument based on the existence of substitution rules. In 1981, N. de Bruijn showed that Penrose tiling also can be obtained by projection of a discrete plane in $\mathbb{R}^5$ (with vertices in $\mathbb{Z}^5$) to the nearest two-dimensional hyperplane. In this talk we show that projection tilings of this type admit (a countable infinity of different) substitution rules if and only if there exists a "quadratic" (partially) hyperbolic lattice automorphism that commutes with the projection. As the latter condition is very easy to verify, we obtain a simple characterization (and many new examples) of such renormalizable projection tilings.

On Algebraic Structure of Neighborhoods of Cellular Automata —*full* and *one-way*—[*]

Hidenosuke Nishio[1][†], Maurice Margenstern[2], Friedrich von Haeseler[3]

[1] Iwakura Miyake-cho 204, Sakyo-ku,
606-0022, Kyoto, Japan
e-mail: YRA05762@nifty.ne.jp

[2] LITA, EA 3097,
UFR MIM, University of Metz,
Île du Saulcy,
57045 Metz, France
e-mail: margens@sciences.univ-metz.fr

[3] KU Leuven, Dep. of Electrical Engineering,
Kasteelpark Arenberg 10,
3001 Leuven, Belgium
e-mail: friedrich.vonHaeseler@esat.kuleuven.ac.be

November 9, 2004

Abstract

Recently, we formulated and analyzed the structure of neighborhoods of cellular automata based on the algebraic tool, where the space is presented by a finitely generated group and the neighborhood relation is defined by a semigroup generated by the neighborhood [5][4]. This paper is a continuation with focus on the *full* and *one-way* neighborhoods.

1 Introduction

A cellular automaton (in short CA) is a uniformly structured information processing system defined on a regular cellular space, which is presented as a Cayley

[*]extended abstract for WTCA'04
[†]Corresponding author

graph of a finitely generated group [6]. Usually, the same finite automaton is placed at every point of the space. On the other hand, the neighborhood (index) specifies, for each point of a space, the extent where the information directly comes from. The neighborhood is also assumed to be spatially uniform. Then, the direct neighbors of a point p are obtained by semigroup (associative) operations of the neighborhood indices to p. Consecutive application of operations gives the whole *neighbors* of p. Usually any cell is assumed to be able to communicate with any other cell of the space, sooner or later. Algebraically it means that the subspace generated by a neighborhood coincides with (*fills*) the space itself. This is the case for the space is 2-dimensional grid and the neighborhood is von Neumann and/or Moore. By the way, a distinctive feature of such neighborhoods is that a cell can communicate with neighboring cells in *any directions*.

In the literature, most authors assume a neighborhood which equals the set of generators of the space itself, as is the case of the von Neumann neighborhood for 2-dimensional space. It is, in a sense, a proper way of research, because most problems that are pertinent to CA can be formulated and solved assuming a von Neumann neighborhood. We should note, however, that E.F.Codd has shown a two state self-reproducing CA with a neighborhood of size 85 [2]. Reducing the number of cell states generally requires a larger neighborhood. Since then, the study of neighborhoods has attracted many authors.

For example, the problem of a restricted communication (a restricted neighborhood) of CA has been investigated by many authors. The first considered restriction is *one-way*. One-way one-dimensional CAs have been extensively investigated for topics as : one-way simulation, computational universality, reversibility, real/linear time language recognition including closure properties and so on. As for higher dimensional CAs, however, the investigations on *one-way* or restricted communication are still not many. For CAs defined on the Cayley graph including $\mathbb{Z}^d$, Z.Roka first showed that a d-dimensional CA with von Neumann neighborhood can be simulated in $d+1$ time by a one-way CA with one-way von Neumann neighborhood [6]. V.Terrier recently discussed closure properties of the rotation of 180° of 2-dimensional languages, when von Neumann and Moore neighborhoods are restricted to *one-way* [7].

On the other hand, we formulated the whole neighborhoods defined for a CA space and discussed the problem whether a neighborhood *fills* the space or not [5]. Obviously, for a cell to communicate with any other cell, the neighborhood must *fill* the space. In particular, for the space $\mathbb{Z}^2$, we proved the neighborhood *3-horse* $\{(2,1),(-2,1),(1,-2)\}$ fills. Also we gave a necessary and sufficient condition for a general 3-horse $\{(a,b),(-a,b),(b,-a)\}$ to fill [4]. In this paper, we will show the generalization to d-dimensional Euclidean space and torus. At the same time we will discuss the *one-way* neighborhood, comparing the previous and the present definitions.

2 Preliminaries

A CA is defined by a 4 tuple (S, N, Q, f), where S is the regular space, at each point of which the same cell is located. S can be infinite or finite. A regular construct is typically represented by a Cayley graph of a finitely generated group. N is the neighborhood (index) which consists of a finite number of neighborhood indices. The same neighborhood is applied to every point of S.

$$N = \{n_1, n_2, ..., n_s\} \subset S$$

The set of states Q is a finite set of symbols. The local map $f : Q^N \rightarrow Q$ gives a local state transition function, which is common to every cell.

The global dynamics of CA is defined as the global map $F : C \rightarrow C$, where elements of $C = Q^S$ are called global configurations. F is uniquely induced from f by

$$F(c)(x) = f(c(xn_1), c(xn_2), \cdots, c(xn_s)),$$

for any $c \in C$ and $x \in S$. When starting from a configuration c, the behavior (trajectory) is given by

$$F^{t+1}(c) = F(F^t(c)),$$

for any $c \in C$ and $t \geq 0$, where $F^0(c) = c$.

2.1 Cellular space S and neighbors relative to S

Here we are interested in the structure of a CA and give an algebraic setup of the space S and neighborhood N.

2.1.1 Space

We assume that $S = \langle G \mid R \rangle$, where $G = \{g_1, g_2, ..., g_r\}$ is a set of finite number of generators (symbols) and R is a finite set of relations (equalities) of words over G and G^{-1}:

$$R = \{w_i = w'_i \mid w_i, w'_i \in (G \cup G^{-1})^*, i = 1, ..., n\} \tag{1}$$

Every element (point) of S is presented as a *word* $x \in (G \cup G^{-1})^*$,
where $G^{-1} = \{g^{-1} | \ g \in G\}$ and $g \cdot g^{-1} = 1$, where 1 is the empty word or the identity if S happens to be a (semi)group. A free group is expressed by $S = \langle G | \ \emptyset \rangle$.

2.1.2 Neighborhood and neighbors

Let a space $S = \langle G \mid R \rangle$ be given. A neighborhood (index) for S is expressed by

$$N = \{n_1, n_2, ..., n_s\} \subset (G \cup G^{-1})^* \tag{2}$$

Now we recursively define the *neighbors* of CA as follows.

(1) Let $p \in S$, then the *1-neighbors* of p, denoted as pN^1, is the set

$$pN^1 = \{pn_1, pn_2, ..., pn_s\}. \tag{3}$$

(2) The $(m+1)$-*neighbors* of p, denoted as pN^{m+1}, are given as

$$pN^{m+1} = pN^m \cdot N, \; m \geq 0, \tag{4}$$

where $pN^0 = \{p\}$.

Note that the computation of pn_i has to comply with the relations R defining $S = \langle G|R \rangle$.

We may say that the information contained in the cells of pN^m reaches the cell p after m time steps. In the sequel, without loss of generality, we principally treat the m-neighbors $1N^m$ (N^m in short) of the origin 1 of S. The cardinality of N^m, denoted as $\#(N^m)$, is called the *neighborhood size.*

(3) ∞-*neighbors* of p, denoted as pN^∞, is defined by

$$pN^\infty = \bigcup_{m=0}^{\infty} pN^m. \tag{5}$$

(4) ∞-neighbors of 1, denoted as N^∞, is called *neighbors* (of CA).

Then we have an algebraic result, which is proved by the fact that the procedure to generate a subsemigroup and the above mentioned recursive definition of N^∞ are the same. For a recursive procedure to generate subalgebras, we refer to page 33 of [1].

Proposition 2.1

$$N^\infty = \langle N \mid R \rangle_{sg}, \tag{6}$$

where $\langle N \mid R \rangle_{sg}$ means the semigroup generated by N with relation R.

Remarks: A subalgebra $\langle A \rangle$ generated by a set A is the smallest subalgebra that contains A. A is called a set of generators. For a subalgebra there can be more than one set of generators. To avoid confusion, the group (res. subgroup) generated by G (res. G') is denoted by $\langle G \mid R \rangle_g$ (res. $\langle G' \mid R \rangle_{sg}$). When R is understood, we simply write as $\langle G \rangle_g$ (res.$\langle G' \rangle_{sg}$). We also use the terms of g-generate and sg-generate. A semigroup is an associative system. In addition we assume here that the cancellation rule holds. Then we have,

Lemma 2.1

$$\langle g_1, g_2, ..., g_r, g_1^{-1}, g_2^{-1}, ..., g_r^{-1}\rangle_{sg} = \langle g_1, g_2, ..., g_r\rangle_g \tag{7}$$

2.1.3 Set of neighborhoods

For a fixed space S, we consider the set of all finite neighborhoods relative to S and denote it as $\mathcal{N}^S$. In $\mathcal{N}^S$, we define special subclasses of neighborhoods which will be studied later.

Definition 2.1 (Symmetric) *If $N = N^{-1}$, then N is called a symmetric neighborhood. (For additive groups $N = -N$.)*

Von Neumann and Moore neighborhoods are symmetric.

Definition 2.2 (One-way) *If $(N \cap N^{-1}) \setminus \{1\} = \emptyset$, then N is called a one-way neighborhood.*

Remarks on the definition: Z. Roka [6] defines *one-way* neighborhoods of p by deleting every edge labelled with generators G of the space. So, the information comes only from $pg^{-1}, g \in G$. In other words, the neighborhood of one-way CA is G^{-1}. The present definition allows a neighborhood N to be one-way, if $N \subset (G \cup G^{-1})^*$ and $(N \cap N^{-1}) \setminus \{1\} = \emptyset$, which is a wider definition than Roka's. V. Terrier [7] treats two one-way neighborhoods; $\mathcal{V}_1 = \{(0,0),(1,0),(0,1)\}$ and $\mathcal{V}_3 = \{(0,0),(1,0),(0,1),(1,1)\}$, both of which generate the NE-quarter of $\mathbb{Z}^2$ as shown in Examples below.

Theorem 4.1 shows that in case of one-way neighborhood N_{3H} any pair of cells in $\mathbb{Z}^2$ can communicate with each other (the time is generally different depending on the direction), which is not true in the case of the ordinary definition of *one-way*, including that of Roka and Terrier. The neighborhood of a 3-horse could be said to be *locally one-way* but *globally* not. The plausibility of this definition is left for future discussion.

2.1.4 Intrinsic neighbors

Define the *intrinsic* m-neighbors, denoted as $[N^m]$, as those cells that can reach the origin in exactly m steps. That is,

$$[N^m] = N^m \setminus N^{m-1}, \tag{8}$$

where $[N^0] = \{1\}$.

Evidently we see,

$$N^\infty = \bigcup_{m=0}^{\infty} [N^m]. \tag{9}$$

2.2 Examples of spaces and neighborhoods

- Set of integers $\mathbb{Z} = \langle a \mid \emptyset \rangle$. Elementary CA has the neighborhood $\{a^{-1}, 1, a\}$.
- $S = \mathbb{Z}_4 = \langle a \mid a^4 = 1 \rangle = \{1, a, a^2, a^3\}$ is a circle consisting of 4 points. If $N_A = \{a\}$, then $N_A^1 = \{a\}, N_A^2 = \{a^2\}, N_A^3 = \{a^3\}$ and $N_A^4 = \{1\}$. $N_A^\infty = S$. If $N_B = \{a^2\}$, then $N_B^2 = \{1\}$. $N_B^\infty = \{1, a^2\}$.
- 2-dimensional rectangular grid: $\mathbb{Z}^2 = \langle a, b \mid ab = ba \rangle$.
  The following examples are given in the additive group presentation with generators $\{(1,0),(0,1)\}$ with relation $\{(1,0)+(0,1) = (0,1)+(1,0)\}$.

  (1) Von Neumann neighborhood $N_V = \{(0,0),(1,0),(0,-1),(-1,0),(0,1)\}$.
  (2) Moore neighborhood $N_M = \{(x,y) | x, y \in \{-1,0,1\}\}$.

  (3) See Fig.1.
  (3-1) Horse $N_H = \{(\pm 2, \pm 1), (\pm 1, \pm 2)\} =$

  $$\{(2,1),(2,-1),(-2,-1),(-2,1),(1,2),(1,-2),(-1,-2),(-1,2)\}.$$

  (3-2) 3-horse $N_{3H} = \{(2,1),(-2,1),(1,-2)\} \subset N_H$.
  (3-3) Keima $N_K = \{(1,2),(-1,2)\} \subset N_H$.

  (4) See Fig.2 (North-East quarter of $\mathbb{Z}^2$) for an *effect* of the origin included by a neighborhood. Many authors tacitly assume the origin (the cell in question) to be included by the neighborhood or treat it as a special argument of the local function.

  One-way neighborhood $N_{NE} = \{(1,0)(0,1)\}$. For $m \geq 1$,

  $$N_{NE}^m = \{(x,y) | x, y \in \mathbb{N}_0, x + y = m\}$$

  Now we include the origin in the neighborhood (5) and have another one-way neighborhood $N_{NE'} = \{(0,0),(1,0)(0,1)\}$. Then,

  $$N_{NE'}^m = \{(x,y) | x, y \in \mathbb{N}_0, x + y \leq m\}.$$

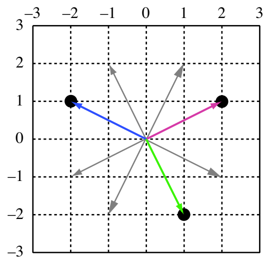
  We easily see,

  $$N_{NE'}^\infty = N_{NE}^\infty,$$

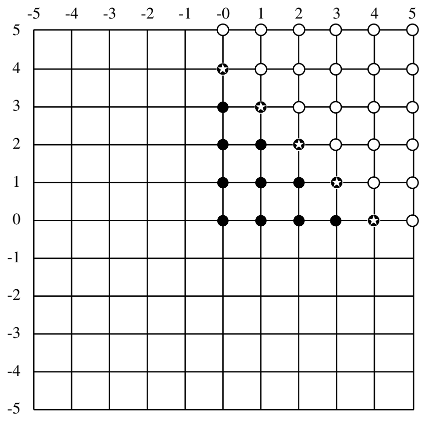
  $$N_{NE}^m = [N_{NE}^m] = [N_{NE'}^m],$$

  and

  $$\#(N_{NE}^m) = m + 1, \quad \#(N_{NE'}^m) = (m+1)(m+2)/2.$$



(a) Fig.1 Horses



(b) Fig.2 N^m_{NE} and N'^m_{NE}

- Torus $\mathbb{Z}_m \times \mathbb{Z}_n = \langle a, b \mid ab = ba, a^m = 1, b^n = 1\rangle$ with $m, n \in \mathbb{N}$ the set of nonnegative integers.
  Additive group with generators $a = (1, 0), b = (0, 1)$ and relations $ma = (0, 0)$ and $nb = (0, 0)$.

- Hexagonal grid $\langle a, b, c \mid a^2 = 1, b^2 = 1, c^2 = 1, (abc)^2 = 1\rangle$.
  Note that $ab \neq ba, ac \neq ca, bc \neq cb$. Since $a = a^{-1}$, any neighborhood is symmetric.

- Triangular grid $\langle a, b, c \mid abc = 1, acb = 1\rangle$.
  The commutativity $ab = ba, ab = ba, bc = cb$ is derived from the relations.

3 Analysis of neighborhoods

In this section some basic properties of neighborhoods are analyzed, assuming a space $S = \langle G|R\rangle$. $p, q, ..$ stand for points of S and $m, m', .. \in \mathbb{N}_0$.

Lemma 3.1 (transitivity) *If $q \in pN^m$ and $r \in qN^{m'}$, then $r \in pN^{m+m'}$.*

Lemma 3.2 (slide) *If $q \in pN^m$ then for any $r \in S$, $qr \in prN^m$. Particularly, if $q \in pN^m$, then $qp^{-1} \in N^m$.*

Proposition 3.1 *If N is a symmetric neighborhood, then for any $p, q \in S$ and nonnegative integer m,*

$$p \in qN^m \Longleftrightarrow q \in pN^m.$$

Corollary 3.1 *If N is a symmetric neighborhood, then for any $p, q \in S$*

$$p \in qN^\infty \Longleftrightarrow q \in pN^\infty.$$

Proposition 3.2
$$\langle N \rangle_g \supseteq N^\infty.$$

Proof : The procedure of generating a group includes the procedure of making m-neighbors for any $m \geq 0$.

Proposition 3.3 *There are neighborhoods N such that $\langle N \rangle_g \supsetneqq N^\infty$.*

Proof: We state some examples of one-way CAs.

(a) $N = \{a\}$. $\langle N \rangle_g = \mathbb{Z}$. But $N^\infty = \mathbb{N}_0$.

(b) Keima $N_K = \{(1,2), (-1,2)\}$, which is a one-way neighborhood, sg-generates a part of the sparse plane but does not fill it. In fact,

$$N_K^\infty = \{(a-b, 2a+2b) \mid a, b \in \mathbb{N}_0\} \subsetneqq \{(a-b, 2a+2b) \mid a, b \in \mathbb{Z}\} = \langle N_K \rangle_g$$

.

Proposition 3.4 *If N is symmetric, then $\langle N \rangle_{sg} = N^\infty$.*

Proof: Since N is symmetric, if N contains g then it does g^{-1}. Therefore, by Proposition 2.1 and Lemma 2.1 we have $\langle N \rangle_g = N^\infty$. The 3-horse provides an example of a non-symmetric neighborhood N such that $\langle N \rangle_{sg} = \mathbb{Z}^2 = N^\infty$.

4 Filling problem of neighborhoods

As for the problem of limited communication of CA, we observed various one-way neighborhoods for the space $\mathbb{Z}^2$, which do not sg-generate the whole space $\mathbb{Z}^2$. In this section, we study on the condition for a neighborhood to *fill* the space.

Definition 4.1 *A neighborhood N is said to fill S if and only if $N^\infty = S$.*

Then, by Proposition 2.1, we have

Proposition 4.1 *N fills S if and only if $\langle N \rangle_{sg} = S$.*

As for a typical non-standard neighborhood, we studied the *horse power problem* and showed the following results [5][4].

Theorem 4.1 *A 3-horse $N_{3H} = \{(2,1), (-2,1), (1,-2)\}$ fills $\mathbb{Z}^2$.*

Note that N_{3H} is not symmetric and *one-way.*

Theorem 4.2 *The generalized 3-horse $H_{G3H} = \{(a,b), (-a,b), (b,-a)\}$ fills $\mathbb{Z}^2$, if and only if the following conditions hold.*

condition 1: $gcd(a,b) = 1$.
condition 2: $a + b = 1 \ mod\ 2$, where $a > b > 0$.

We have generalized the problem to d-dimensional space and proved the following theorem using the theory of integral matrices [3].

Theorem 4.3 *Let $x_1, ..., x_s \in \mathbb{Z}^d$, where $s \geq d+1$. Then the neighborhood $\{x_1, ..., x_s\}$ fills the space or $\langle x_1, ..., x_s \rangle_{sg} = \mathbb{Z}^d$, if and only if the following two conditions hold.*

condition 1: $\gcd(\{det([x_{i_1}, ..., x_{i_d}])|i_1, ..., i_d \in \{1, ..., s\}\}) = 1$.
condition 2: $\underline{0} \in int(conv(\{x_1, ..., x_s\})$. *(The zero of $\mathbb{R}^d$ should be in the interior of the convex hull of $\{x_1, ..., x_s\}$.)*

The theory of integer matrices also allows to state necessary and sufficient conditions for a horse to fill the torus.

Theorem 4.4 *If the horse moves on a d-dimensional torus $T = \mathbb{Z}_{m_1} \times ... \times \mathbb{Z}_{m_d}$, with $m_i \in \mathbb{N}$ and if the horse's move are $\{x_1, ..., x_s\} \subset \mathbb{Z}^2$, then the horse fills the torus T if and only if*

$$\gcd(\{det([y_1, ..., y_d])|y_i \in \{x_1, ..., x_s, m_1 e_1, ..., m_d e_d\}) = 1,$$

where e_i is the i-th unit vector.

5 Concluding remarks

Decision problems whether a neighborhood fills or not have been treated before [4], but we have not entered the problem here. The conditions stated in Theorem 4.3 could be tested in polynomial time in $s \times d$. Future research: As is defined in Section 2.1.3, considering the class $\mathcal{N}^S$ of all finite neighborhoods for a space S will lead to a new unified theory of CAs. For instance, assume a fixed space S and a local function f_s with s arguments, then the set N_s of all neighborhoods $\{N \in \mathcal{N}^S | \#(N) = s\}$ of cardinality s seems useful for the research of *variable communication CAs*. What happens if the neighborhood is changed for a fixed local function ? Which neighborhood of N_s is the *best* one for f_s ?

This research was done during the first author's stay at Faculty of Informatics, University of Karlsruhe, September-October 2004. Many thanks are due.

References

[1] Burris, S., Sankappanavar, H. P.: *A Course in Universal Algebra*, The millennium edition, Open website, 2000.

[2] Codd, E. F.: *Cellular Automata*, Academic Press, 1968.

[3] Newman, M.: *Integral Matricies*, Academic Press, 1972.

[4] Nishio, H., Margenstern, M.: An algebraic Analysis of Neighborhoods of Cellular Automata, Submitted 2004.

[5] Nishio, H., Margenstern, M.: *An algebraic Analysis of Neighborhoods of Cellular Automata*, Technical Report 1375, RIMS, Kyoto University, May 2004, Proceedings of LA Symposium, Feb. 2004.

[6] Roka, Z.: One-way cellular automtata on Cayley graphs, *Theoretical Computer Science*, **132**, 1994, 259–290.

[7] Terrier, V.: *Cellular automata recognizer with restricted communication*, Technical Report 32, Turku Center for Computer Science, June 2004, Proceedings of DMCS'04.

State-Efficient 1-Bit-Communication Solutions for Some Classical Cellular Automata Problems

Hiroshi Umeo[1]*, Masaru Kanazawa[1], Koshi Michisaka[1] and Naoki Kamikawa[1]

[1] Univ. of Osaka Electro-Communication
Faculty of Informatics
Neyagawa-shi, Hatsu-cho, 18-8, Osaka, 572-8530, Japan

September 10, 2004

Abstract

We propose several state-efficient 1-bit-communication algorithms for some classical cellular automata problems. A 1-bit inter-cell communication model ($\mathrm{CA}_{1-\mathrm{bit}}$) studied in this paper is a subclass of cellular automata (CA) whose inter-cell communication is restricted to 1-bit. We study a sequence generation problem, a firing squad synchronization problem and an early bird problem, all of which are known as the classical and fundamental problems in cellular automata.

First we consider the sequence generation problem. It is shown that there exists a 1-state $\mathrm{CA}_{1-\mathrm{bit}}$ that can generate in real-time a context-sensitive sequence such that $\{2^n \mid n = 1, 2, 3, ...\}$. Secondary, we study the firing squad synchronization problem on two-dimensional $\mathrm{CA}_{1-\mathrm{bit}}$. We give a two-dimensional $\mathrm{CA}_{1-\mathrm{bit}}$ which can synchronize any $n \times n$ square and $m \times n$ rectangular arrays in $2n - 1$ and $m + n + \max(m, n)$ steps, respectively. In addition, we propose a generalized synchronization algorithm that operates in linear time on two-dimensional rectangular arrays with the general located at an arbitrary position of the array. The time complexities for the first two algorithms developed are one to two steps larger than optimum ones proposed for O(1)-bit communication model. In the last, we give a 1-bit implementation for an early bird problem. It is shown that there exists a 19-state $\mathrm{CA}_{1-\mathrm{bit}}$ that solves the early bird problem in linear time.

*Corresponding author: umeo@cyt.osakac.ac.jp

1 Introduction

In recent years cellular automata (CA) have been establishing increasing interests in the study of modeling real phenomena occurring in biology, chemistry, ecology, economy, geology, mechanical engineering, medicine, physics, sociology, public traffic, etc. Cellular automata are considered to be a good model of complex systems in which an infinite one-dimensional array of finite state machines (cells) updates itself in synchronous manner according to a uniform local rule.

In this paper, we study a sequence generation problem [1, 4, 7, 19, 20], a firing squad synchronization problem [2, 5, 6, 10-13, 15-17, 21-25] and an early bird problem [3, 8, 9, 14, 23], all of which are known as the classical and fundamental problems studied extensively on O(1)-bit communication models of cellular automata. An O(1)-bit communication model is a conventional CA where the amount of communication bits exchanged at one step between neighboring cells is assumed to be O(1)-bit, however, such bit-information exchanged between inter-cells has been hidden behind the definition of conventional automata-theoretic finite state descriptions. On the other hand, a 1-bit inter-cell communication model studied in this paper is a new CA whose inter-cell communication is restricted to 1-bit. We call the model 1-bit CA in short, and it is denoted as $\mathrm{CA}_{1-\mathrm{bit}}$. The number of internal states of the 1-bit CA is assumed to be finite in a usual way. The next state of each cell is determined by the present state of itself and two binary 1-bit inputs from its left and right neighbor cells. Thus the 1-bit CA can be thought to be one of the most powerless and simplest models in a variety of CAs.

In the next section 2, we define formally a 1-bit communication cellular automaton ($\mathrm{CA}_{1-\mathrm{bit}}$) and give a computational relation between the conventional CA and the $\mathrm{CA}_{1-\mathrm{bit}}$. In section 3, we consider a sequence generation problem on $\mathrm{CA}_{1-\mathrm{bit}}$ and give several non-regular sequences that can be generated in real-time by $\mathrm{CA}_{1-\mathrm{bit}}$. In section 4, a synchronization problem is studied and three 1-bit implementations of synchronization algorithms for two-dimensional square and rectangular arrays will be given. In the last section, an early bird problem is considered and an efficient 19-state implementation will be given. Due to the space available, we omit the details of the proofs of theorems given below.

2 One-Bit Communication Cellular Automata

A one-dimensional 1-bit inter-cell communication cellular automaton [13, 18-20] consists of an infinite array of identical finite state automata, each located at positive integer point. Each automaton is referred to as a cell. A cell at point i is denoted by C_i where $i \geq 1$. Each C_i, except C_1, is connected with its left and right neighbor cells via a left or right one-way communication link, where those communication links are indicated by right- and left-going arrows, as is shown in Fig. 1, respectively. Each one-way communication link can transmit only one bit at each step in each direction. One distinguished leftmost cell C_1,

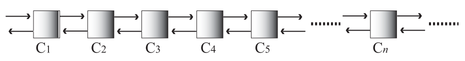


Figure 1: One-dimensional cellular automaton having 1-bit inter-cell communication links.

the communication cell, is connected to outside world.

A cellular automaton with 1-bit inter-cell communication (abbreviated by $\mathrm{CA}_{1-\mathrm{bit}}$) consists of an infinite array of finite state automaton $A = (Q, \delta)$, where

1. Q is a finite set of internal states.

2. δ is a function, defining the next state of any cell and its binary outputs to its left and right neighbor cells, such that δ: $Q \times \{0,1\} \times \{0,1\} \rightarrow Q \times \{0,1\} \times \{0,1\}$, where $\delta(p,x,y) = (q,x',y')$, p, $q \in Q$, $x, x', y, y' \in \{0,1\}$, has the following meaning: We assume that at step t the cell C_i is in state p and receiving binary inputs x and y from its left and right communication links, respectively. Then, at the next step $t+1$, C_i assumes state q and outputs x' and y' to its left and right communication links, respectively. Note that binary inputs to C_i at step t are also outputs of C_{i-1} and C_{i+1} at step t. A quiescent state $q \in Q$ has a property such that $\delta(q,0,0) = (q,0,0)$.

Thus the $\mathrm{CA}_{1-\mathrm{bit}}$ is a special subclass of *normal* (i.e., *conventional*) cellular automata studied so far. Let N be any normal cellular automaton with a set of states Q and a transition function $\delta : Q^3 \rightarrow Q$. The state of each cell on N depends on previous states of itself and its nearest neighbor cells. This means that the total information exchanged per one step between neighboring cells is O(1)-bit. By encoding each state in Q with a binary sequence of length $\lceil \log_2 |Q| \rceil$, sending the sequences sequentially bit by bit in each direction via each one-way communication link, receiving them bit by bit again, and decoding them into their corresponding states in Q, the $\mathrm{CA}_{1-\mathrm{bit}}$ can simulate one step of N in $\lceil \log_2 |Q| \rceil$ steps. This observation gives the following computational relation between the normal CA and $\mathrm{CA}_{1-\mathrm{bit}}$.

Theorem 2.1 [19] *Let N be any normal cellular automaton with time complexity $T(n)$. Then, there exists a $CA_{1-\mathrm{bit}}$ which can simulate N in $kT(n)$ steps, where k is a positive constant integer such that $k = \lceil \log_2 |Q| \rceil$ and Q is the set of N's states.*

3 Real-time Sequence Generation Problem on $\mathbf{CA}_{1-\mathrm{bit}}$

Now we define the sequence generation problem on $\mathrm{CA}_{1-\mathrm{bit}}$. Let M be a $\mathrm{CA}_{1-\mathrm{bit}}$ and $\{t_n |\, n = 1, 2, 3, ...\}$ be an infinite monotonically increasing positive integer sequence defined on natural numbers such that $t_n \geq n$ for any $n \geq 1$. We have a semi-infinite array of cells, shown in Fig. 1, and all cells, except C_1, are in quiescent state and output 0 to their left and right communication links at time $t = 0$. The communication cell C_1 assumes a special state in Q and outputs 1 to its right communication link at time $t = 0$ for an initiation of the sequence generator. We say M generates a sequence $\{t_n | \, n = 1, 2, 3 ...\}$ in k *linear-time* if and only if the left end cell of M outputs 1 to the outside world via its left communication link at time $t = kt_n$, where k is a fixed positive integer. We call M *real-time* generator, when $k = 1$.

The 1-bit CA can be thought to be one of the most powerless and simplest models in a variety of CAs. In spite of its simplicity, the $\mathrm{CA}_{1-\mathrm{bit}}$ can generate a variety of context-sensitive sequences given below.

Theorem 3.1 [19] *There exists a 3-state $CA_{1-\mathrm{bit}}$ that can generate $\{n^2 | \, n = 1, 2, 3, ...\}$ in real-time.*

Theorem 3.2 [19] *There exists a 9-state $CA_{1-\mathrm{bit}}$ that can generate Fibonacci sequence in real-time.*

Theorem 3.3 [20] *Prime sequence can be generated in real-time by a 34-state $CA_{1-\mathrm{bit}}$.*

A class of 1-state $\mathrm{CA}_{1-\mathrm{bit}}$ is the simplest one in $\mathrm{CA}_{1-\mathrm{bit}}$. We show that there exists a context-sensitive sequence that can be generated in real-time by a 1-state $\mathrm{CA}_{1-\mathrm{bit}}$. The context-sensitive sequence is such that $\{2^n | \, n = 1, 2, 3, ...\}$. A transition rule set for the $\mathrm{CA}_{1-\mathrm{bit}}$ M that generates $\{2^n | \, n = 1, 2, 3, ...\}$ in real-time is as follows: $M = \{Q, \delta\}$, where $Q = \{a, q\}, \delta(a, 0, 0) = (a, 0, 0), \delta(a, 0, 1) = (a, 1, 0), \delta(q, 0, 0) = (q, 0, 0), \delta(q, 0, 1) = (q, 1, 1), \delta(q, 1, 0) = (q, 1, 1)$ and $\delta(q, 1, 1) = (q, 0, 0)$. The leftmost cell C_1 always assumes a state a and $\mathrm{C}_i (i \geq 2)$ takes a state q at any step. In Fig. 2, we show some snapshots for the real-time generation of the sequence. Small right and left black triangles, ▶ and ◀, shown in the figure, indicate a 1-bit signal transfer in the right or left direction between neighbor cells. A symbol in a cell shows its internal state.

Theorem 3.4 *An infinite sequence $\{2^n | \, n = 1, 2, 3, ...\}$ can be generated in real-time by a 1-state $CA_{1-\mathrm{bit}}$.*

4 Firing Squad Synchronization Problem on $\mathbf{CA}_{1-\mathrm{bit}}$

In this section, we study a famous firing squad synchronization problem on the newly introduced 1-bit CA model for which solution gives a finite-state protocol for synchronizing a large scale of cellular automata. The problem was originally proposed by J. Myhill to synchronize all parts of self-reproducing cellular automata [12]. The firing squad synchronization problem has been studied extensively in more than 40 years [2, 5, 6, 10-13, 15-17, 21-25].

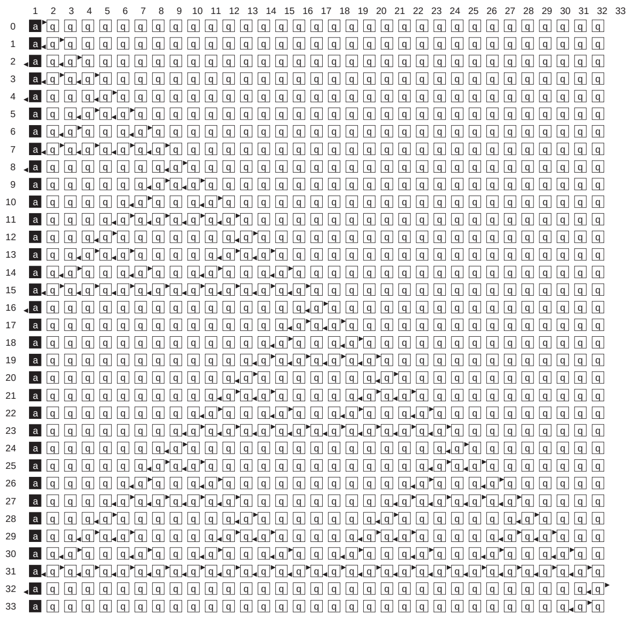


Figure 2: Snapshots for real-time generation of $\{2^n |\, n = 1, 2, 3, ...\}$ on a 1-state $\mathrm{CA}_{1-\mathrm{bit}}$.

4.1 Synchronization Algorithm on 1-D Array

Before presenting our synchronization algorithms on 2-D $\mathrm{CA}_{1-\mathrm{bit}}$, we review two algorithms for synchronizing 1-D $\mathrm{CA}_{1-\mathrm{bit}}$ with the general at the left end or at an arbitrary position of the array. Nishimura, Sogabe and Umeo [13] designed an optimum-step firing squad synchronization algorithm on $\mathrm{CA}_{1-\mathrm{bit}}$, where $2n - 2$ steps are required for synchronizing n cells on 1-D array and the general is located at the left end of the array. The algorithm, that is referred to as NSU algorithm, is stated as follows:

Theorem 4.1 [13] *There exists a $CA_{1-\mathrm{bit}}$ which can synchronize n cells with the general on the left end in $2n - 2$ steps. The $CA_{1-\mathrm{bit}}$ constructed has 78 internal states and 208 transition rules.*

Theorem 4.2 given below is a generalized version of Theorem 4.1.

Theorem 4.2 [22] *There exists a $CA_{1-\mathrm{bit}}$ which can synchronize n cells in $n+max(k, n - k + 1)$ steps, where k is any integer such that $1 \leq k \leq n$ and a general is located on the kth cell from the left end of the array. The total number of internal states and transition rules of the $CA_{1-\mathrm{bit}}$ realized on a computer is 282 and 721, respectively.*

We develop some synchronization algorithms for 2-D 1-bit inter-cell communication CA models. Fig. 3 shows a finite two-dimensional cellular array

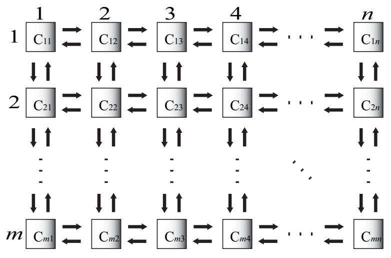


Figure 3: Two-dimensional cellular automaton.

consisting of $m \times n$ cells. A cell on (i, j) is denoted by $\mathrm{C}_{i,j}$. Each cell is an identical (except the border cells) finite state automaton. The array operates in lock-step mode in such a way that the next state of each cell (except border cells) is determined by both its own present state and the present binary inputs from its north, south, east and west neighbors. All cells, except the general cell, are initially in the quiescent state with the property that the next state of a quiescent cell with four 0 inputs is the quiescent state again and outputs 0 to its four neighbors. Given an array of $m \times n$ identical cellular automata, including a *general* on the $\mathrm{C}_{1,1}$ cell which is activated at time $t = 0$, we want to give the description (state set and next-state function) of the automata so that, *at some future time*, all the cells will *simultaneously* and, *for the first time*, enter a special *firing* state. The set of states must be independent of m and n. The tricky part of the problem is that the same kind of soldier with a fixed number of states is required to synchronize, regardless of the size m and n of the array. Several 2-D synchronization algorithms and their implementations have been presented in Shinar [15] and Szwerinski [16] for O(1)-bit communication models.

4.2 Synchronization Algorithm on Square Arrays

We present a new synchronization algorithm that runs in $(2n-1)$ steps on $n \times n$ square arrays. Our algorithm is one step slower than that of Shinahr [15] for O(1)-bit communication model and operates as follows. By dividing the entire square array into n L-shaped 1-D arrays such that the length of the ith L is $2n - 2i + 1$ $(1 \leq i \leq n)$, we treat the square firing as n independent 1-D firings with the general located at the center cell. On the ith L, a general is generated at $C_{i,i}$ at time $t = 2i - 1$, and the general initiates the horizontal and vertical firings on the row and column arrays. In our construction, we apply the previous NSU algorithm [13] for each row and column firing. The array fires in optimum time $t = 2i - 1 + 2(n - i + 1) - 2 = 2n - 1$. We have tested our transition rule set on squares of size 2×2 to 1000×1000. The total number of internal states and transition rules of the $\mathrm{CA}_{1-\mathrm{bit}}$ realized on a computer is 127 and 405, respectively. Figure 4 shows snapshots of configurations of our 127-state

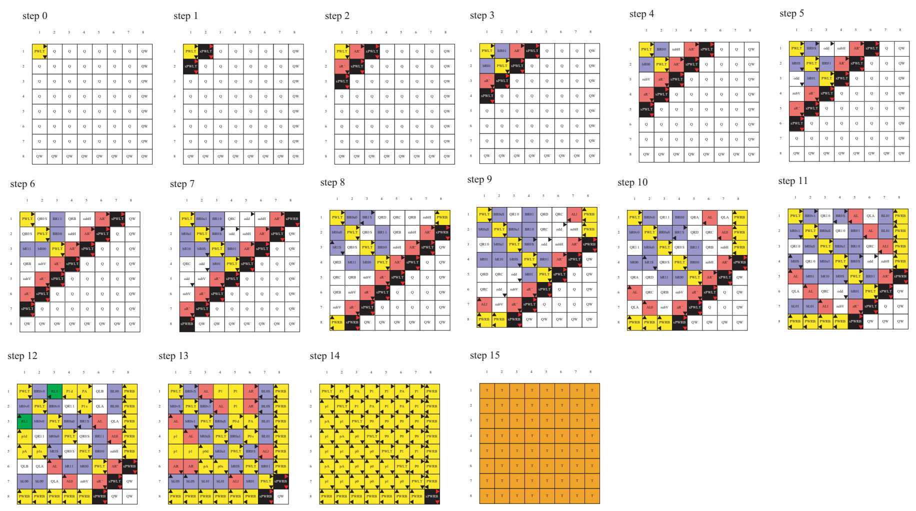


Figure 4: Snapshots of the $(2n-1)$-step square firing squad synchronization algorithm with the general on the north west corner.

synchronization algorithm running on a square of size 8×8. Thus we have:

Theorem 4.3 *There exists a 2-D $CA_{1-\mathrm{bit}}$ which can synchronize $n \times n$ cells in $2n-1$ steps.*

4.3 Synchronization Algorithm on Rectangular Arrays

The generalized firing squad synchronization algorithm presented in [Theorem 6] can be applied to the problem of synchronizing rectangular arrays with the general at the north-west corner. The configuration of the generalized firing on 1-D arrays can be mapped on 2-D array. The rectangular array is regarded as $\min(m, n)$ L-shaped 1-D arrays, where they are synchronized independently using the generalized firing squad synchronization algorithm. We have implemented the algorithm on a computer. In Fig. 5, we show snapshots of the synchronization process on 5×8 rectangular array. The total number of internal states and transition rules of the $\mathrm{CA}_{1-\mathrm{bit}}$ realized on a computer are 862 and 2217, respectively. Thus we have:

Theorem 4.4 *There exists a 2-D $CA_{1-\mathrm{bit}}$ which can synchronize $m \times n$ rectangular arrays in $m + n +$ max(m, n) steps.*

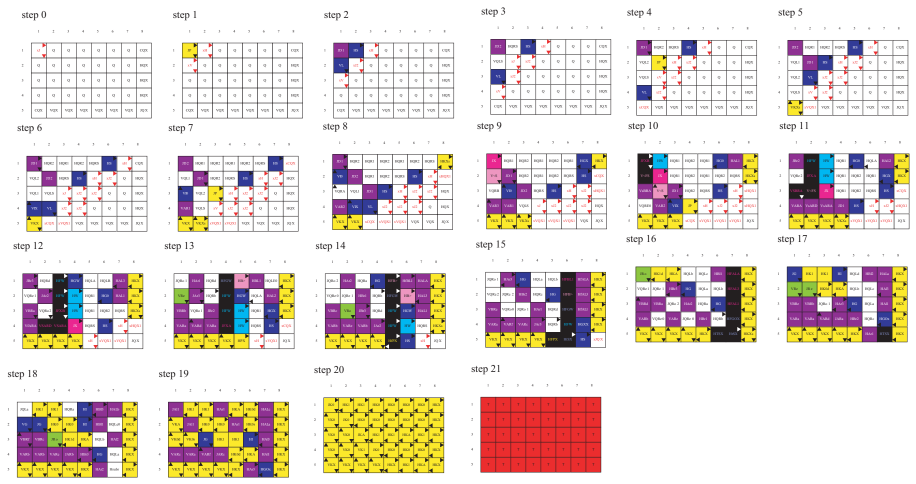


Figure 5: Snapshots of our rectangular firing squad synchronization algorithm with the general at the north-west corner.

4.4 Generalized Synchronization Algorithm on 2-D Rectangular Arrays

In this subsection, we study the generalized synchronization algorithm on rectangular arrays. Let r, s be any integer such that $1 \leq r \leq m$, $1 \leq s \leq n$. At time $t = 0$ the general cell $\mathrm{C}_{r,s}$ is in *fire-when-ready* state that is an initiation signal to the array. Before presenting the 1-bit algorithm, we show a simple and efficient mapping scheme developed for O(1)-bit CA model that embeds any generalized one-dimensional synchronization algorithms onto two-dimensional arrays [21]. Now we consider a 2-D array of size $m \times n$. We divide mn cells into $m + n - 1$ groups g_k, $1 \leq k \leq m + n - 1$, defined as follows;

$$g_k = \{C_{i,j} | (i-1) + (j-1) = k - 1\}.$$

That is,
$g_1 = \{C_{1,1}\}$, $g_2 = \{C_{1,2}, C_{2,1}\}$, $g_3 = \{C_{1,3}, C_{2,2}, C_{3,1}\}$, . . . , $g_{m+n-1} = \{C_{m,n}\}$.

Let M be any one-dimensional $\mathrm{CA}_{1-\mathrm{bit}}$ that fires ℓ cells in $T(\ell, k)$ steps, where the general is on C_k and k be any integer such that $1 \leq k \leq \ell$. We assume that M has $m + n - 1$ cells. We consider the one-to-one correspondence between the ith group g_i and the ith cell C_i on M such that $g_i \leftrightarrow C_i$, where $1 \leq i \leq m + n - 1$. We can construct a 2-D $\mathrm{CA}_{1-\mathrm{bit}}$ N so that all cells in g_i simulates the ith cell C_i in real-time and N can fire any $m \times n$ arrays with the general $\mathrm{C}_{r,s}$ at time $t = T(m+n-1, r+s-1)$ if and only if M fires any 1-D arrays of length $m+n-1$ with the general on C_{r+s-1} at time $t = T(m+n-1, r+s-1)$.

Based on the generalized 1-D algorithm given in [Theorem 4.2], we get the following 2-D generalized synchronization algorithm that fires in $T(m, n, r, s)$

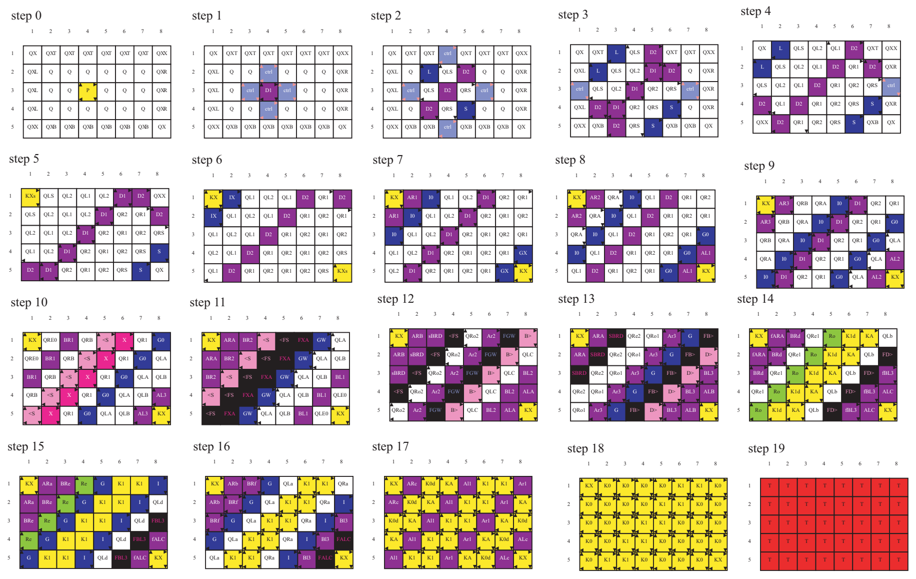


Figure 6: Snapshots of our generalized rectangular firing squad synchronization algorithm operating on an array of size 5×8 with the general on $C_{3,4}$.

steps given below. The total number of internal states and transition rules of the $CA_{1-\mathrm{bit}}$ realized on a computer is 300 and 2333, respectively. In Fig. 6, we show snapshots of the 300-state generalized synchronization algorithm running on rectangular array of size 5×8 with the general on $C_{3,4}$. Thus we have:

Theorem 4.5 *There exists a 2-D 1-bit communication $CA_{1-\mathrm{bit}}$ that can synchronize any $m \times n$ rectangular arrays in $T(m,n,r,s)$ steps, where (r,s) is an arbitrary initial position of the general and $T(m,n,r,s)$ is defined as follows: $T(m,n,r,s) = m + n + \max(r+s, m+n-r-s+2) + O(1)$.*

Szwerinski [16] proposed an optimum-time generalized 2-D firing algorithm with 25600 internal states that fires any $m \times n$ array in $m+n+\max(m,n)-\min(r, m-r+1)-\min(s, n-s+1)-1$ steps. Our 2-D generalized synchronization algorithm is relatively larger than the optimum one proposed by Szwerinski [16], however, the number of internal states required for the firing is the smallest known at present.

5 Early Bird Problem on $CA_{1-\mathrm{bit}}$

In this section we study an early bird problem on $CA_{1-\mathrm{bit}}$. Consider a one-dimensional CA consisting of n cells in which any cell initially in quiescent state may be excited from the outside world. The problem is to give a description

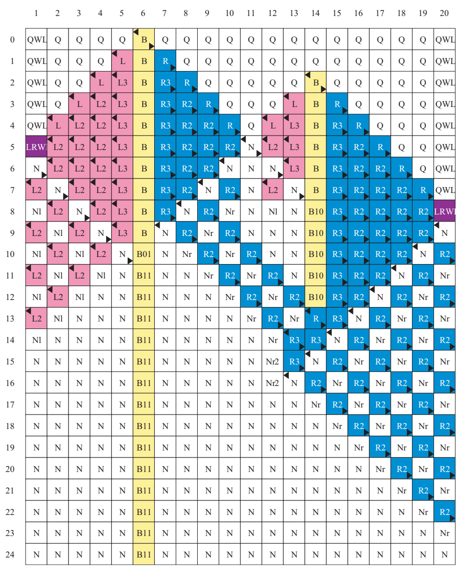


Figure 7: Snapshots of a 19-state implementation of the early bird problem on $\mathrm{CA}_{1-\mathrm{bit}}$.

(state set and next state function) of the automata so that the first excitation(s) can be distinguished from the later ones. The problem was originally devised by Rosenstiehl, Fiksel and Holliger [14] to design some graph-theoretic algorithms operating on networks of finite state automata. Büning [3] showed that a 5-state solution developed by Legendi and Katona [8] is an optimum one in the number of states on O(1)-bit communication model. We have got a 19-state implementation on $\mathrm{CA}_{1-\mathrm{bit}}$ that operates in $3n$+O(1) steps for one-dimensional $\mathrm{CA}_{1-\mathrm{bit}}$ of size n. In Fig. 7, we show some snapshots of the 19-state implementation.

Theorem 5.1 *Ther exists a 19-state $CA_{1-\mathrm{bit}}$ that can solve the early bird problem in $3n$+$O(1)$ steps.*

6 Conclusion

A sequence generation problem, a firing squad synchronization problem and an early bird problem are known as the classical and fundamental problems which have been studied extensively on O(1)-bit communication models of cellular automata. In this paper, we have developed several optimum algorithms for those problems and given their efficient implementations on $\mathrm{CA}_{1-\mathrm{bit}}$. It has been shown that there exists a context-sensitive sequence that can be generated in real-time by a 1-state $\mathrm{CA}_{1-\mathrm{bit}}$. We have proposed several new synchronization algorithms for two-dimensional $\mathrm{CA}_{1-\mathrm{bit}}$ and implemented them on a computer.

Most of the algorithms proposed are one to two steps larger than optimum ones proposed for O(1)-bit communication model. We are convinced that there still exist interesting new synchronization algorithms, although more than 40 years have passed since the development of the problem. A 19-state implementation on $\mathrm{CA}_{1-\mathrm{bit}}$ is also given for the early bird problem.

References

[1] M. Arisawa: On the generation of integer series by the one-dimensional iterative arrays of finite state machines (in Japanese). *The Trans. of IECE* 71/8, Vol. 54-C, No.8, pp. 759-766, (1971).

[2] R. Balzer: An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, vol. 10(1967), pp. 22-42.

[3] H. K. Büning: The early bird problem is unsolvable in a one-dimensional cellular space with 4 states. *Acta Cybernetica*, vol. 6(1983), pp.23-31.

[4] P. C. Fischer: Generation of primes by a one-dimensional real-time iterative array. *J. of ACM*, Vol., 12, No.3, pp. 388-394, (1965).

[5] E. Goto: A minimal time solution of the firing squad problem. Dittoed course notes for Applied Mathematics 298, Harvard University, (1962), pp. 52-59.

[6] M. Hisaoka, H. Yamada, M. Maeda, T. Worsch, and H. Umeo: A design of firing squad synchronization algorithms for a multi-general problem and their implementations (in Japanese). *Technical Report of IEICE NLP2002-133* (2003), 103-108.

[7] I. Korec: Real-time generation of primes by a one-dimensional cellular automaton with 11 states. *Proc. of 22nd Intern. Symp. on MFCS '97, Lecture Notes in Computer Science*, 1295, pp. 358-367, (1997).

[8] T. Legendi and E. Katona: A 5-state solution of the early bird problem in a one-dimensional cellular space. *Acta Cybernetica*, Vol.5, No.2, pp. 173-179, (1981).

[9] T. Legendi and E. Katona: A solution of the early bird problem in an n-dimensional cellular space. *Acta Cybernetica*, Vol.7, No.1, pp. 81-87, (1984).

[10] J. Mazoyer: A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, vol. 50(1987), pp. 183-238.

[11] J. Mazoyer: On optimal solutions to the firing squad synchronization problem. *Theoretical Computer Science*, vol. 168(1996), pp. 367-404.

[12] E. F. Moore: The firing squad synchronization problem. in *Sequential Machines, Selected Papers* (E. F. Moore ed.), Addison-Wesley, Reading MA., (1964), pp. 213-214.

[13] J. Nishimura, T. Sogabe and H. Umeo: A Design of Optimum-Time Firing Squad Synchronization Algorithm on 1-Bit Cellular Automaton. *Proc. of The 8th International Symposium on Artificial Life and Robotics*, pp. 381-386, (2003).

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

[15] I. Shinahr: Two- and three-dimensional firing squad synchronization problems. *Information and Control*, vol. 24(1974), pp. 163-180.

[16] H. Szwerinski: Time-optimum solution of the firing-squad-synchronization-problem for n-dimensional rectangles with the general at an arbitrary position. *Theoretical Computer Science*, vol. 19(1982), pp. 305-320.

[17] S. L. Torre, M. Napoli and M. Parente: A compositional approach to synchronize two dimensional networks of processors. *Theoretical Informatics and Applications*, 34 (2000) pp. 549-564.

[18] H. Umeo: Linear-time recognition of connectivity of binary images on 1-bit inter-cell communication cellular automaton. *Parallel Computing*, 27, pp. 587-599, (2001).

[19] H. Umeo and N. Kamikawa: A design of real-time non-regular sequence generation algorithms and their implementations on cellular automata with 1-bit inter-cell communications. *Fundamenta Informaticae*, 52 (2002) 255-275.

[20] H. Umeo and N. Kamikawa: An infinite prime sequence can be generated in real-time by 1-bit inter-cell communication cellular automaton. *Preproc. of The 6th International Conference on Developments in Language Theory*, Univ. of Kyoto Sangyo,(2002), pp.372-382.

[21] H. Umeo, M. Maeda and N. Fujiwara: An efficient mapping scheme for embedding any one-dimensional firing squad synchronization algorithm onto two-dimensional arrays. *Proc. of the 5th International Conference on Cellular Automata for Research and Industry*, LNCS 2493, Springer-Verlag, pp.69-81(2000).

[22] H. Umeo, M. Hisaoka, K. Michisaka, K. Nishioka and Masashi Maeda: Some new generalized synchronization algorithms and their implementations for large scale cellular automata. *Proc. of The Third International Conference on Unconventional Models of Computation*(C. S. Calude, M. J. Dinneen and F. Pepper Eds.), 2002, pp.276-286.

[23] R. Vollmar: On two modified problems of synchronization in cellular automata. *Acta Cybernetica*, Vol.3, No.4, pp. 293-300, (1978).

[24] R. Vollmar: Algorithmen in Zellelarautomaten. Teubner, pp.192, Stuttgart, (1979).

[25] A. Waksman: An optimum solution to the firing squad synchronization problem. *Information and Control*, vol. 9(1966), pp. 66-78.

Self–reproducing self–assembling evolutionary automata*

Jiří Wiedermann
Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
e–mail jiri.wiedermann@cs.cas.cz

September 13, 2004

Abstract

We introduce a computational model of a so–called globular universe which represents generalization of both classical cellular automata and contemporary models of self–assembly. Similarly as the latter mentioned model our model utilizes a multiset of globules which are endowed by self–organizing ability controlled by a finite state mechanism; these computational units are not fixed in a predetermined structure. The environment abounds in these units which are available at places where needed for a self–assembly of various objects. Within a globular universe we define the notion of self–reproducing evolutionary automaton. This notion refers to an automaton being at the beginning of a lineage of self–reproducing automata which leads to self–reproducing automata with arbitrary complex finite state control mechanisms via a series of mutations of intermediate automata. The ideas presented in this paper complement von Neumann's results on self–reproducing automata in a static universe by offering a precise definition of what is meant by "evolutionary self–reproduction" and by designing a dynamic nondeterministic universe with a self–reproducing self–assembling evolutionary automaton.

1 Introduction

It seems that all experience of mankind points to the fact that machines can produce only simpler machines than the original ones. The only exception have been living beings if considered as machines: not only can an organism produce an almost genuine copy of itself but, as Darwin has shown, in a long run, in an evolutionary process these "machines" keep improving. It was von Neumann who in late 1940 started a systematic quest for a logical, rather than material,

*This research was partially supported by grant No. 1ET100300419 within the National Research Program "Information Society".

basis of biological self–reproduction. He first proposed a mechanistic model. It consisted of a "robot" operating in a sea of its own spare parts. The robot had some elementary functions for moving around, identifying and collecting the required parts and assembling them together and possessed a tape with instructions for building a copy of itself by making use of these elementary functions. After constructing a replica of itself, the robot finally copied its instruction tape and inserted it into the replicated robot which could then start the same activities. By this design, it is generally agreed that von Neumann discovered basic principles for the process of biological self–reproduction. Namely, there had to be a program, instruction sequence to be used in two different ways: (1) to be interpreted as instructions for constructing an offspring, and (2) to be copied passively, without being interpreted. Quite understandingly, von Neumann was not able to construct a working model of his mechanistic self–reproducing automaton which would represent a convincing proof of soundness of his design idea. However, in 1953, following Stanislaw Ulam's vision of cellular automata, he invented a cellular automaton implementation of his mechanistic model. His cellular "robot" made use of a cellular automaton with 29 states per cell, consisted of approximatively 200 000 cells and its description took more than 200 pages [6]. Thus the topic of self–reproduction entered the field of the automata theory and since then cellular automata have been used in numerous applications and variations. Interestingly, in the contemporary automata theory within the computer science, only a limited attention has been paid to the self–reproducing issue. Nevertheless, this issue has migrated into the field of artificial life where it continues to be a subject of a vivid development. In this context, especially two questions are of interest. First, what exactly was the problem to which von Neumann gave his answer? And second, how good this answer was?

As far as the first question is concerned, von Neumann did not formulate the problem of self–reproduction for cellular automata in such a way that it would show that his solution is a satisfactory and complete answer to that problem. This is because constructing an automaton, which is merely able to produce its own copy, is interesting especially in a real world but less in the general framework of cellular automata. Namely, as pointed out by several researchers (cf. [3] for an extensive discussion of this problem), each cell of a cellular automaton can "reproduce itself" by properly initializing a cell in its immediate neighborhood. This cell can even posses universal computational properties, i.e., it can act as a motile processor of a single–tape universal Turing automaton working over a linear array of cells holding the input data. Thus, universality alone is not a good enough reason for devising a self–reproducing cellular automaton of an immense complexity as von Neumann did. Perhaps, von Neumann was aware of this since in his *Theory of Self–Reproducing Automata* ([6], p. 92) he asked: *"Can the construction of automata by automata progress from simpler types to increasingly complicated types? Also, assuming some suitable definition of "efficiency", can this evolution go from less efficient to more efficient automata?"* Nowadays it appears that the final aim of von Neumann's design was not the universality but the "self–improvement" issue of a self–reproducing automaton which, however, was not settled at all by von Neumann. As far as the second question is concerned, in the past there were several trials to refine the von Neumann design of a self–reproducing automaton (cf. [3]). Nevertheless, in the absence of a precise task definition it has been difficult to compare the alternative solutions w.r.t. the given task and to judge in what sense the alternative solutions were better. In these efforts an evolution has still not been the main issue and that is why several authors, inclusively e.g. Herman [1] and recently also McMullin [3] called for a mathematical definition of the "evolutionary growth of complexity" (as it is called in [3]) of self–reproducing automata.

In this paper we formulate a new mathematical model of self–assemblage which is a generalization of both classical cellular automata and contemporary models of self–assemblage. This model allows to introduce nondeterminism into the self–assemblage process. Within this model we give a mathematical definition of a self–reproducing evolutionary automaton which captures essential aimes of von Neumann's design: the requirement of self–reproduction and the possibility of evolutionary changes leading *"from simpler types to increasingly complicated types",* as von Neumann put it. In our definition, the notion of "efficiency" of an automaton is measured by the minimal number of states of an equivalent finite control mechanism. Finally we describe nondeterministic globular universe and a self–reproducing self–assembling evolutionary automaton existing within this universe. This automaton is conceptually much simpler than von Neumann's proposal and includes a "built–in" nondeterministic evolutionary mechanism which leads directly to the infinite evolutionary lineages of automata with increasingly complex finite state control.

The structure of the paper is as follows: in Section 2 the globular universe is defined. Section 3 shows the computational equivalency of the model with cellular automata and Turing machines. Next Section 4 defines the notion of a self–reproducing evolutionary machine. Finally, Section 5 describes the design of a nondeterministic globular universe in which a self–reproducing evolutionary globular automaton exists. Conclusions in Section 6 mention possible avenues for a further research.

2 Globular universe

The basic particles possessing latent self–assembly properties in which our universe abounds and from which all our ensembles will be constructed are computational units called *globules*. As their name suggests globules have a shape of tiny balls. Each globule can be seen as an "embodied" finite automaton that can find itself in one of the finite number of states. These states determine the self–assembly properties of the globules, i.e., their abilities to bind with other globules. The globules move freely in an empty 3D space — except of the globules and their complexes there is nothing else in the space. They are subject to a random motion, occasionally colliding and possibly binding one with each other. The behavior of globules on this occasion is determined by their state at that moment and is described by an *interaction function*. This function determines new states of the globules and their behavior after their interaction. Prior to giving a formal definition which abstracts the previously given informal view of a globular universe we will introduce few preliminaries.

Let $\mathcal{S}$ be a geometrical sphere of a unit radius in a 3D Euclidean space. A *contact domain*, or simply a *contact* on the surface of $\mathcal{S}$ is defined as the interior of any closed convex curve on its surface; the curve itself is a part of the contact domain. Each contact domain has a uniquely determined position on the surface w.r.t. the given coordinate system. We will consider only such contact domains whose boundaries are computable — i.e. there is a Turing machine algorithm that will "draw" the domain at the required position on the surface of $\mathcal{S}$. An example of a contact domain is a single point on a sphere, or a "belt" drawn around the "equator" of a sphere, a spherical circle drawn at a sphere's pole, etc. We will consider spheres with a finite number of contact domains on their surface; domains can overlap. Each domain is endowed by certain properties which are characterized by the color of the domain, and by the affinity of the domain. We will see that the domains represent the mechanism underlying the self–assembly properties of globules. We define the respective notions more formally.

Definition 2.1 *A* globular configuration space *is the set* $C = \{D_1, \ldots, D_k, \Sigma, \Gamma, \pi\}$, *where*

- $D_1, \ldots, D_k$, *for* $k > 0$, *are* contact domains, *or* contacts *defined on the surface of a unit geometrical sphere;*
- Σ *is the finite alphabet of* contact domain colors*;*
- $\Gamma = \{neutral, join\}$ *is the set of* affinities [1]*;*
- $\pi : \{D_1, \ldots, D_k\} \to 2^{\Sigma \times \Gamma}$ *is a function that for each contact domain defines the admissible set of* contact properties.

Any element of $(\Sigma \times \Gamma)^k = \Omega$ *is called a* configuration *(or a* state*, respectively) of a globule from space* C.

The state of a globule g can change when g interacts with another globule h. We say that g interacts with h if and only if both globules come into a contact, or if they are already in a contact — one with the other and one of the two changes its state. If more than two globules interact simultaneously this parallel multiple interaction is broken into a randomly ordered sequential series of pairwise interactions. The configuration changes, during an interaction, are governed by a so–called *interaction function* $\Phi : \Omega^2 \to \Omega^2$. Let $c_g, c_h \in \Omega$ be the states of g and h at the time of their interaction. Then $\Phi(c_g, c_h) = (c'_g, c'_h)$ is to be read as *"after the interaction of g with h, the state of g changes from c_g to c'_g whereas the state of h changes from c_h to c'_h"*.

[1] We can also consider a richer set of affinities, e.g. of form $\Gamma = \{neutral, attract, join, repel\}$ or affinities of a variable strength, given by an integer number; then the priorities among the affinities must be stated similarly as in the case of a self–assembly model introduced in [4].

At the interaction times the behavior of the globules is governed by the domain colors and the affinities. Consider two contact domains D_i and D_j with $\pi(D_i), \pi(D_j) \subseteq \Sigma \times \Gamma$ and two globules $g, h \in C$. Let $(\sigma_g, \gamma_g) \in \pi(D_i)$ and $(\sigma_h, \gamma_h) \in \pi(D_j)$ be the contact properties of the respective globules at these contacts immediately *after* the interaction, i.e., after the application of Φ. When both colors and affinities match, i.e., $\sigma_g = \sigma_h$ and $\gamma_h = \gamma_h = \gamma$ then the globules will behave as follows:

- if $\gamma = \mathit{neutral}$, then g and h are neither attracted to nor repelled from each other via contacts D_i and D_j; that is, both globules will not be joined via the respective contacts;
- if $\gamma = \mathit{join}$, then g and h will join one with each other via a *bond* established between the contacts D_i and D_j.

In all other cases the globules will behave as if both had a neutral affinity.

A bond between two globules can only be established when the globules have not been already bonded to other globules in a position which prevents both globules to enter a position allowing touching the contact domains needed for a new bond. If a new bond can be established then both globules stabilize in a relative position satisfying the needs of all existing bonds. If there are more possibilities for such a positioning, one of them is chosen randomly.

Note that if contact domains are not point domains (i.e., if they have a positive area), then they can make a "movable" bond between any points within their contact areas. This gives some freedom to globules which can for instance self–assemble into an elastic structure with a variable curvature. For instance, globules with an attracting contact domain in form of a belt around their equators can self–organize into a ball–shaped structure of a certain minimal and maximal radius.

Summarizing formally our previous notions, we arrive at the following definition of a globular universe:

Definition 2.2 A globular universe $\mathcal{U} = (C, \Phi)$ *is a multiset of globules and their ensembles with globules from the globular configuration space* C *interacting via the interaction function* Φ.

Note that for some arguments Φ need not be defined. That is, in fact Φ determines what kind of globules may interact in $\mathcal{U}$ and whether and how the properties of globules will change on this occasion. For instance, it need not be the case that the state of a globule can be changed into any other state or that an originally neutral globule will ever bind to some other globule via some sequence of interactions.

Next we will aim towards a definition of an evolution within a globular universe. The idea is to see the evolution as a series of "snapshots" taken at interaction times and documenting in this way the interactions of objects within the universe. In the sequel we will consider only globular universes with a finite number of finite globular objects, with a potentially infinite supply of globules. The objects can consist of globules from the entire configuration space. However, the configuration space of the supplied globules may be restricted. Such globules will be said to arrive from the *environment* while the objects under consideration will be said to find themselves in an *observable part* of the universe. Once arriving from the environment and interacting with an object within the observable universe a globule will become a part of this universe unless stated otherwise. The globules arriving from the environment into the observable universe will also be seen as an *input* into the observable universe.

A *universe configuration* reflects the static aspects of a situation in the observable universe at an interaction time. These aspects include the description of all objects within the universe and their spatial relations. The dynamic aspects, i.e., the movements of objects leading to their collisions and state changes of globules are captured through the interactions among the objects.

A globule represents a basic object. All other globular objects will essentially be self–assembled objects formed by multisets of globules. In general, a *description of an object* O is given by an adjacency graph where its nodes correspond to the globules and its edges to the bonds among the globules. Of course, the mapping between the nodes and the globules and that between the edges and the bonds must be included in this definition. However, in many cases we will deal with objects having a regular structure leading to their simpler representation. In such a case an object O will be defined by

- its *size*, i.e., No. of elements in the underlying multiset from which it is composed;
- the subset $S \subseteq \Omega$ of admissible states of globules potentially creating that multiset;
- its spatial organization which is described as a computable invariant (predicate) that holds for all globules in O and captures their adjacency relations with their neighbors, i.e. it captures in fact bonds among the respective globules (see examples of globular objects in the sequel).

The *spatial relations* among the globular objects are described with the help of predicates which hold for all globules within the objects. Such a predicate could be e.g. a binary predicate TOUCHES(A,B) with the meaning *"object A touches object B".* Other predicates could be e.g. OUTSIDE, INSIDE, NEXT_TO and the like. In fact, all our predicates that will be used in the sequel will be computable in a polynomial time w.r.t. the size of the objects involved.

Let $\mathcal{E} \subseteq \mathcal{U}$ be the multiset of globules in the environment, O_t be the set of all objects in the observable part of $\mathcal{U}$ at an interaction time t, and let R_t be the set of spatial relations holding among the objects in O_t at that time. The *(observable) universe configuration* $C_t = \{O_t, R_t\}$ at time t is described by the description of all globular objects in the universe and relations among them at time t. Let INTERACT_t be the set of all interactions to be realized over objects at time t. This set is given by the list of pairs of interacting globules at that time. To each globule its state and membership in $\mathcal{E}$ or to an object in O_t must be stated. Thus, an interaction can occur either between two globules from the environment or between a globule from the environment and another one from an object, or between two globules within the same object or within different objects. The evolution starts at time $t = 0$ in a certain initial universe configuration C_0. The set of interactions $\mathrm{INTERACT}_0$ is then applied to the objects in O_0 to get C_1. In this way the evolution proceeds by applying interactions to objects within configurations at interaction times t_i, for $i = 1, 2, \ldots$ Note that due to the fact that dynamic information about globular objects is not captured in a configuration and the states of globules arriving from $\mathcal{E}$ are in general unknown beforehand, the interactions among the objects cannot be computed (e.g. by using Newton laws) and thus the entire evolution cannot be determined from knowing the initial configuration only (as it is the case with a classical cellular automaton). Under such a scenario an evolution is seen as an on–line interaction between the objects and elements from $\mathcal{E}$ (if any) representing the input into the observable part at that time. As a result, applying $\mathrm{INTERACT}_t$ onto objects in C_t we get C_{t+1}. Also note that by transiting from C_t to C_{t+1} not only objects but also their spatial relations may change.

Next we will describe basic globular objects which we will need in the sequel.

A globule in a given state presents the simplest object. Interactions among globules lead to emergence of more complex objects. For instance, after a collision of two globules, g and h, respectively, a pair $g.h$ can emerge if the interaction function is such that it results in joining g to h.

A simple globular object with a regular rigid structure is a *grid*. It is a two–dimensional rectangular array with globules in identical initial states residing in the array's cells. Each globule has 4 neighbors (in the north, south, east, and west direction) to which it is bonded once for all times and with which it can interact and change their state.

A useful globular object is a *strand*. It is a linear string of globules concatenated via bonds. For each non–empty strand its first globule is defined and for each globule in a non–empty strand, except the last one, its successor is defined. The important operation over strands is an operation of extending a strand by a globule; this globule is added behind the last globule of the strand. Another operation is a copy operation; this operation will be described in Section 4. A strand with both ends joined together is called a *ring*.

So far we considered in essence a deterministic globular universe: thanks to the deterministic definition of the interaction function, from a given configuration and the set of interactions to be performed over the objects described by that configuration at that moment, the next configuration is computable in a unique way (the notable exception could be "random serialization" of multiple collisions). It is obvious that similarly as in the case of classical cellular automata probabilism could also be introduced into the model (e.g. in order to model "mutations"), and also non–determinism. As we will see later, the latter option is particularly convenient because it allows considering situations, which are in principle possible without bothering much about their probabilities. In this case, instead of an interaction function, we will consider a *nondeterministic* *interaction relation* giving a finite number of possible outcomes for each interacting pair of globules. In our considerations we will then always say explicitly which "branch" of a nondeterministic development is to be used. In analogy with the standard nondeterminism known from the automata theory, we will consider that branch which will lead "where we need", i.e., in the case of self–reproducing automata to the self–reproduction of the automaton at hand, in the case of evolving automata their evolution, etc.

Similarly, we will also consider nondeterministic universes in the following sense: if the spatial constellation of existing objects leaves an access path free for a globule to come from the environment, a globule in a required state will "come flying" from the environment if available in $\mathcal{E}$. That is, from among all globules, which can in principle arrive, the one "we need" will arrive, indeed. This is a similar condition as considered, e.g., in the tile assembly model where tiles are available when and where needed. A *nondeterministic universe* will be a universe with a nondeterministic interaction relation and nondeterministic arrival of globules. Considering such universes enables concentrating on self–assembly aspects of interactions.

Within any globular universe, by interaction of globules various complexes of self–assembled globules can arise. We will be particularly interested in self–replicating assemblies arising from certain initial universe configurations in certain universes. Prior to submerging into the related problems we will briefly study the power of our model.

3 Globular Universe, Cellular Automata and Self–assemblage

First of all, we show how one–dimensional cellular automata can be simulated within a globular universe. For such a purpose we use the grid structure described in the previous section. The respective globules will have four bonding contacts spread equidistantly along their equators and each globule will simulate one cell of our cellular automaton. It is obvious that starting from the "seed" globules containing the input to the cellular automaton, any cellular automaton can be simulated by such a grid.

The question of the reverse simulation of a globular universe on a cellular automaton is a little more complicated. W.l.o.g. we can consider a Turing machine in place of a cellular automaton. For a Turing machine it is possible to keep on its tapes the representation of observable universe configurations and to realize the respective operations needed for updating these configurations. To do this in the finite control of a Turing machine at hand the complete table describing the interaction function of the simulated cellular automaton must be stored. The list of interactions to be performed over the objects represented on the machine's tapes will appear at the machine's input after processing the previous list of such operations. In this way the simulation can proceed as needed.

As far as simulations of models of self–assembly are concerned, the situation is as follows. There seems to be no "referential" model of self–assemblage. Therefore we will concentrate onto the elementary model, so–called tile assembly model considered, e.g., in [4]. This model consists of tiles which possess pre–defined self–assembly properties. These properties are described with the help of interaction strength assigned to the sides of rectangular sides; the sides bind when the total interaction strength exceeds a given parameter. This parameter corresponds to the affinity strength in our model (see the footnote in the previous chapter) and therefore can be simulated by our model when using globules with four contact points equidistantly placed along the globule's equator instead of the square–shaped tiles. It is clear that the globular universe presents generalization of the tile assembly model.

4 Self–reproducing evolving automata

For the definiteness of our subsequent discussion we will first define the notion of a self–reproducing automaton in a globular nondeterministic universe. In what follows we will always consider universe $\mathcal{U}$ with an observable part $\mathcal{P} \subseteq \mathcal{U}$ with the environment $\mathcal{E} \subseteq \mathcal{U}$.

Definition 4.1 *A globular object M in $\mathcal{P}$ is* self–reproducible *if and only if there exists an evolution, starting in a configuration C_M containing M as a single object which, after carrying a finite number of interactions among the globules from M and $\mathcal{E}$ gives rise to a configuration with at least two occurrences of M in $\mathcal{P}$.*

The notion of an offspring of a self–reproducing automaton is defined in an obvious way:

Definition 4.2 *We say that a self–reproducing object M_1 is an* ancestor *of a self–reproducing object M_2 (or that M_2 is an* offspring *of M_2) if and only if there is an evolution from a configuration C_{M_1} containing exactly M_1 into a configuration C_{M_2} containing M_2.*

Obviously, thanks to the nondeterminism an offspring of a self–reproducing automaton M can be either an exact copy of M or a different automaton (if different pathes in the evolution have been taken). It may happen that the different offspring is still a self–reproducing machine. If this new automaton is in a sense better than the old automaton we have embarked on a specific "positive" evolution. In order to capture in what sense the new automaton could be better we turn our attention towards the information processing ability of the underlying machine. In the sequel we will consider automata controlled by a finite state mechanism which is "made up" from globules. We will call such class of automata *globular automata*. Obviously, the processing power of a finite state mechanism is directly related to the number of its states. In principle, automata with more states are able to distinguish among a greater number of different situations and thus are able to generate a richer repertoire of actions which can lead to a more sophisticated behavior and/or to a different machine's architecture. Yet in concrete cases it may happen that an automaton has more states than needed in order to generate a given behavior. Thus, the number of states alone is not an adequate indicator of an automaton's power. Therefore the complexity of a globular automaton should be defined by the number of states of a minimal finite state mechanism controlling an automaton in an identical way. Fortunately, in the sequel we will not need to know the minimal automaton producing a given behavior; instead, we will show that in principle such an automaton can be constructed in the process of evolution.

Definition 4.3 *We say that a self–reproducing globular automaton M is a* self–reproducing evolving automaton *(or that M* has an evolutionary potential*) in $\mathcal{U}$ if and only if for any given finite state automaton* A *among the offsprings of M in $\mathcal{U}$ there is a self–reproducing evolutionary globular automaton with a control mechanism realizing* A.

Obviously, a similar definition of a self–reproducing evolving automaton can be given also for the case of classical cellular automata; however, a realization of such an automaton need not be as simple as is the realization of such an automaton in a nondeterministic globular universe.

5 Constructing a self–assembly evolutionary self–reproducing globular automaton

The existence of a self–reproducing globular automaton within a given universe depends much on the properties of the globules in $\mathcal{U}$. For instance, in a universe with single–state globules, no self–reproducing automata can exist. Similarly, in universes with only neutral globules (i.e., no self–assembly is possible) no complex objects can be built and therefore only single–globule self–replication objects can exist in it, with no evolutionary potential.

The case where self–assembly works and the environment supplies globules "as we like it" is more interesting.

Theorem 5.1 *There exists a nondeterministic globular universe with a self–reproducing evolving globular automaton.*

A sketch of the proof: Let $\mathcal{U} = (C, \Phi)$ be a nondeterministic globular universe. We will define explicitly neither its configuration space nor the respective interaction function; rather this will become clear from the course of the proof. Let M be the globular automaton we are after. M will behave as follows: it will react to the sequence of environmental changes which follow a certain regular pattern described by a regular language $R \subseteq \Omega^*$ recognized by M. In such a case, i.e., if and only if M recognizes a word $w \in R$, M will either self–reproduce or generate a self–reproducing offspring with a different behavior, possibly with more states in its control mechanism than M had.

Let $w = \sigma_1\sigma_2\ldots\sigma_k \in \Omega^*$ be a word. This word will be "presented" to M as a series of "waves", the first wave consisting of globules from environment $\mathcal{E} \subseteq \mathcal{U}$ in state σ_1, etc. A "wave" means a situation when all globules arriving from the environment are in state σ_i, for $i = 1, 2, \ldots, k$. In this case, we can imagine that as though has M floated in a sea of globules in state σ_1, then σ_2, etc. and these globules interact with all globules on M's surface. We say that w is an input to M and that M *accepts* w if and only if M on input w will generate either an equivalent offspring or a self–reproducible offspring with a modified finite control mechanism (M must be able to generate either of the two, not always an equivalent offspring).

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite state automaton recognizing R. First we show how the transition function of this automaton will be represented in M. For simplicity of explanation, assume first that the cardinality of both Σ and Q is substantially less than the cardinality of C. Under such assumption there is a 1 to 1 correspondence between Q and a subset of C and between Σ and a subset of C and there are sufficiently many globules which can be used for other purposes rather than for representing sets Q and Σ. Now we can represent each element of these sets by a corresponding globule from C. In the sequel we will not distinguish between the elements of these sets and their globular representation. That is, we in fact assume that $\Sigma, Q \subset C$. Moreover, instead of saying "a globule in state q" we will often say "a globule q".

Especially note that there is a distinguished globule in state $q_0 \in C$. Then $\delta : Q \times \Sigma \rightarrow 2^Q$ can be represented as a finite sequence of the segments of globules of form $\$p\$\sigma\$q$ with $p, q \in Q$ and $\sigma \in \Sigma$. Each segment represents a transition of A of form $\delta(p, \sigma) = q$. Of course, for such a representation we need a further globule representing the separator \$. Obviously, since δ is a nondeterministic relation in its representation as stated before, there can be transitions with the same left–hand side. Assuming that the respective globules have strong bonds on their poles the entire δ can be represented as a linear strand of globules with the given syntax. W.l.o.g. we can join the ends of this strand to form a ring. To simplify the explanation we will call the first occurrence of Q in the segment $\$Q\$\Sigma\$Q\$$ as the "current state" whereas the second occurrence as the "new state". For technical reasons we will add to our representation of δ a transition of form $\delta(q_0, b) = q_0$, with $b \notin \Sigma$ and $b \notin \mathcal{E}$ (i.e., a globule in state b will never appear at the input to M).

In order to enable a smooth working of M we will further assume that the basic structure of M is created by a double–ring: in parallel, and bonded to the just described ring there exists a second auxiliary ring. This ring consists of segments of form \$a\$b\$a\$, with $a \notin Q, b \notin \Sigma$. The \$ symbols match in both rings and thus the globules in state a match the states from Q and the globules in state b match elements from Σ in the segment representation of δ in the original ring. This second ring is bonded to the first ring via bonds between the corresponding globules. The original ring will be called the first track whereas the auxiliary ring the second track.

Next M has to remember the current state $q \in Q$ of A. To represent the current state of A we will mark the right–hand side of the transition rule whose application has caused A to enter state q. The marking will be realized in the second track (under the new state in the respective segment in the first track) by changing the state of the corresponding globule to state $s \notin Q$.

Now we need to define the input mechanism to our globular representation of A. In accordance with the assumption that the globules will appear at places when we need and where we need we will simply assume that the input globules in state $\sigma_i \in \Sigma$ will interact with all globules in the ring but only globules b in track two will react to the input and change their states to σ_i.

Now we are in a position to describe one move of a globular automaton.

The situation before the move is as follows: under each globule from Σ in the first track there is a globule $x = \sigma_i \in \Sigma$ corresponding in the second track to the symbol read by A at that time, under each globule, representing a state from Q in each segment, there is an auxiliary globule in state a. The only exception is in exactly one segment where under the second occurrence of a state from Q there is globule s marking the current state of A. Initially, s is placed under the second occurrence of q_0 in the "artificially added" transition $\delta(q_0, b) = q_0$.

Next we must first distribute the information on the current state of A into all segments. To this end s will interact with q and change its state to q. Then it will interact with its left neighbor in order to propagate q to the next segment until the whole ring gets circumvented. After this action under the first two symbols in each segment there is a pair (q, x) denoting the current state and symbol read by A. Then again the whole ring is circumvented by a signal to see if there is a match between the pair (q, x) and the globules above it. All matches get marked by setting the states of globules under the new state in the auxiliary ring to a distinguished state $m \notin Q$.

If there is no match discovered, i.e., in the case that the computation of A has stuck in a configuration from which there is no continuation, the initial state q_0 is entered and the recognition of a new sequence of inputs will begin.

If there is a match, then we have identified the set of transitions that can be potentially applied in this configuration of A. To apply a transition we non-deterministically select one from among all marked transitions and mark it by setting the state under the new state globule to state s. Then we "reset" all globules in the second track, except the one in state s, to their initial values a or b, respectively. Resetting is done by sending a signal around the ring and changing the states of globules m to a and those of globules x to b. In the next move, we assume that the globules from the previous input wave (i.e., globules "read" by the globular automaton) have left the observable universe. If the state marked by s is not a final state from set F the globular automaton is ready for the next move.

Otherwise, M enters the initial state of A, stops recognition and starts the self–replication. The replication proceeds as follows. It starts in the segment with the initial state. Therefrom a special signal is sent moving around the double–ring and doubling its structure. The new ring is built from the incoming globules by transcribing the states of the globules from the original ring into the new ring. The new ring touches the old one only at the currently copied segment. Of course, the newly copied parts do not remain attached to the old ring — the respective bonds get cancelled. The new ring "grows" by inserting further segments to it. The whole process looks as though the new ring "rolled" over the old one and grew on this occasion. If the copying is done faithfully, the result of this process will be an exact replica that is separated from the old ring. However, the copying process can also be carried out nondeterministically, with alternatives introducing changes into the segments of the replicated ring.

In the sequel we will describe the design of a *nondeterministic evolutionary mechanism* which will generate an offspring with a modified transition function of the underlying finite state automaton A. These changes will not be completely arbitrary, e.g. they will preserve the chosen syntax of the transition relation. The new transition relation can either have the same number of segments, or more segments than the original relation.

The case with a transition relation having the same number of segments is easy — it is enough to introduce nondeterministic alternatives into the process of transcribing the elements of δ from the original to its copy. The copying process can be even designed so as to nondeterministically decide to skip copying of certain segments. To remain consistent with the idea of two rings rolling one on each other, the simplest way to achieve the desired effect of skipping one segment is to copy a *junk segment* (i.e. one which could never be interpreted due to its parameters) into the place of the skipped segment. Note that such a process can lead to a "backward" evolution, in a sense that an off-spring can be generated with a transition relation which might have been already generated in a previous generation.

The case when the new transition relation will have more segments then the original one is realized by letting the copying mechanism nondeterministically decide to copy certain segments twice. This is done as follows: after copying a given segment, the copying site (a signal, in fact) can change its direction and instead of proceeding further and carrying out the copying process it will backtrack one segment without carrying the copying process, but freing the already copied segment and binding the previous one in the original back to its counterpart in the copy. As a result, the copied ring roles backwards on the original ring by one segment. Then again the direction of the copying signal is reversed and the copying process can be resumed. Note that by backtracking both in the original and in the copied segment, the newly copied segment (and indeed, all subsequent segments) will be copied *before* the already copied segment, but this makes no harm since our representation of the transition relation is insensitive to the order of segments in it. On the occasion of its second copying the copied segment can be modified. It is clear that the new individuum can recognize a different regular language and can, but need not be able to self–replicate.

Now it should be clear that *over a fixed set* Q *and* Σ the evolutionary mechanism just described can generate arbitrary transition functions for A. That is, we still cannot "enumerate" all finite state control mechanisms (over all sets of states and all alphabets). In order to do so we must relax our assumption on the size of cardinalities of Q, Σ and C. However, if the cardinalities of both Σ and Q are larger than the cardinality of C we cannot map elements of Q and Σ to elements of C as before. Then we must encode elements of Σ and Q in a unary notation into strings of identical globules from C. This will complicate the design of the self–replicating automaton but the above mentioned ideas will work also in this case. Especially, the evolutionary mechanism will now be free to generate "new" states and "new" symbols for A and this will enable evolutions leading to arbitrary complex transition relations. In this case M will become a real self–replicating evolutionary globular automaton according to Definition 4.3. This ends the sketch of the proof of theorem 5.1.

After presenting the construction of the previous self–reproducing evolutionary automaton two remarks are in order.

The first remark concerns the properties of universes in which self–reproducing evolutionary globular automata cannot exist. Namely, it is clear that our construction cannot be realized in case the cardinality of C is too small to allow encodings needed for a globular automaton to work as envisaged. Thus it seems that there is a lower bound on the cardinality of globular configuration space below which no self–reproducing evolutionary automata can exist even if a self–assembly within the respective universe is possible. Moreover, even if a configuration space of a given universe is sufficiently large, there still need not exist transitions among the states of globules which would allow constructing a self–reproducing evolutionary automaton.

The second remark concerns further properties of self–reproducing evolutionary automata which we did not pay attention to in our design. It appears that our self-replicating automaton can easily be made *individuated* in the following sense: arbitrary offsprings of a self–reproducing evolutionary automaton will maintain their separate identities also in a direct contact. This can be achieved by designing the universe and constructing M in such a way that the globules from which the globular automata are made bind only with the globules from the environment. This condition for automata to be individuated has been suggested by McMullin in [2].

6 Conclusion

In the paper we presented three main achievements. The first one was the design of a globular universe which enabled a study of self–assembly in a more general setting than the previous models did. This has been mainly due to getting rid of the rigid structure of cellular space (as in the case of classical cellular automata) or of the necessity to deal with explicit dynamic aspects of particle motion (as in the case of lattice gas automata). The introduction of nondeterminism both into the self–assembly process and into the dynamic input appearance in the observable part of the globular universe enabled concentration to principal existential questions related to the self–assembly of globular objects. A view of a basically continuous evolutionary process as that of a finite series of configurations taken at interaction times and related via sets of on–line interactions enabled a formal treatment of evolution much in the spirit of the computational theory. An application of ideas from the computational complexity theory has led to the second achievement, viz the formal definition of a self–reproducing evolutionary automaton. Last but not least, the third achievement was the design of a specific nondeterministic globular universe and a constructive proof of the existence of a self–reproducing evolutionary automaton within this universe. The automaton itself is substantially simpler than the automata designed by other authors. Moreover, our automaton includes a nondeterministic evolutionary mechanism which guarantees the existence of evolutionary paths towards more complex automata thus answering positively von Neumann's question from the introduction in a constructive way.

We believe that the framework of a globular universe, or a similar one, will enable a further, more detailed study of questions related to self–reproduction, evolution and to the self–emergence of self–reproducing evolutionary machines. As far as the latter machines are concerned, machines with "more complicated" evolving bodies would be of interest. A possible avenue would be to consider the ideas from theoretical biology related to artificial life synthesis (cf.[5]). Here cell-like systems controlled by a genom and embedded in a membrane are considered. The first ideas along these lines in the spirit of our modelling have been sketched in [7]; in fact, the current paper emerged as a result of an effort to bring more formalism into the respective research.

References

[1] Herman, G. T.: On universal computer constructor. Information Processing Letters, Vol 2, pp. 61–64, 1973

[2] McMullin, B.: Some remarks on autocatalysis and autopoiesis. Annals of the New York Academy of Sciences, Vol. 901, pp. 163–174, 2000

[3] McMullin, B.: John von Neumann and the Evolutionary Growth of Complexity: Looking Backwards, Looking Forwards... Artificial Life, Vol 6. Issue 4, Fall 2000, pp. 347-361

[4] Rothemund, P., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). In Proceedings of the thirty-second annual ACM symposium on Theory of computing, pages 459-468. ACM Press, 2000.

[5] Szostak, J. W., Bartel, D. P., Luisi, P. L.: Synthesizing Life. Nature 409 (2001), pp. 389-390.

[6] von Neumann, J.: Theory of Selfreproducing Automata. A. Burks (Ed.), University of Illinois Press, Urbana and London, 1966

[7] Wiedermann, J.: Coupling computational and non–computational processes: minimal artificial life. Pre–proceedings of the Fifth Workshop on Membrane Computing (WMC5), G. Mauri, Gh. Paun, C. Zandroni (Eds.), Dept. of Comp. Sci., University of Milan — Bicocca, Italy, June 16–19, 2004, 444 p.