Complex Systems **7** (1993) 347–365

Mechanisms for Pattern Generation

Karel Culik II
Department of Computer Science,
University of South Carolina, Columbia, SC 29208, USA

Jarkko Kari
Academy of Finland
and
Mathematics Department,
University of Turku, 20500 Turku, Finland

Abstract. Three mechanisms—cellular automata, finite automata, and L-systems—for generating static patterns are compared. Matrix substitution systems, nondeterministic extensions of iterative matrix homomorphisms, are also introduced and shown to be equivalent to finite automata. Two different ways for taking the limit of a sequence of finite resolution patterns produced by any of the mechanisms are studied: one gives an infinite resolution pattern on the unit square, the other one a pattern of infinite size.

1. Introduction

We define the notion of a multiresolution (static) pattern as a coloring of an arbitrarily large subsquare of the tessellation of the plane or in general of an arbitrarily large hypercube of n-dimensional space. As a limit we can consider either a coloring of the tessellation of the infinite plane or of the unit square with infinite resolution. We consider several mechanisms for generating the multiresolution pattern. The common feature is that the same (local) rules define a pattern at an arbitrarily large resolution. This assumes that a pattern has a certain fixed descriptive complexity independent of the resolution used to present it.

First we consider cellular automata (CAs). CAs give low-level models of natural systems; for example, a CA cell can represent a molecule in physics. The universality of CAs implies that every recursive pattern can be generated by a CA. We show examples of extremely simple CAs that produce interesting patterns, such as models of simple crystals.

Second, we discuss finite automata [2] and show that they are equivalent to matrix substitution systems that are nondeterministic extensions of iterated matrix homomorphisms from [9]. In [5] it was shown that (using our

terminology) every pattern specified by a finite automaton can be generated by a CA in linear time. Since every pattern can be approximated by a pattern defined by a finite automaton [3] it follows that an approximation of every pattern can be generated by a CA in linear time. Such an approximation can be effectively found, and an inference program for this purpose is described in [3].

In the last section we discuss how to use the "turtle" interpretation of L-systems [7]. They are a higher-level tool, particularly suitable for modeling the growth of plants [8]. We show that, somewhat surprisingly, every finite automaton (matrix substitution system) can be simulated efficiently by an L-system.

2. Patterns: definitions and notations

To simplify notations we consider only two-dimensional patterns. Generalizations for other dimensions are straightforward.

Definition 1. Let C be a finite set of colors. Frequently $C = \{$'black', 'white'$\}$, but patterns with more colors can be considered as well.

(i) A *finite pattern* $P = (n, f)$ is defined by a positive integer n and a mapping

$$f : \{0, 1, \ldots, n-1\} \times \{0, 1, \ldots, n-1\} \to C.$$

Mapping f assigns colors to the squares of a finite checkerboard of size $n \times n$.

(ii) A *multiresolution pattern* R is defined by an infinite sequence $P_1, P_2, \ldots$ of finite patterns $P_i = (n_i, f_i)$, where $n_1 < n_2 < \cdots$ is an increasing sequence of sizes.

(iii) An *infinite pattern* P_∞ is a mapping

$$\mathbb{Z}^2 \to C$$

that assigns colors to squares of an infinite square tessellation of the plane.

(iv) Let us denote by $I = [0, 1]$ the unit interval of real numbers. An *infinite resolution* (black-and-white) *pattern* R_∞ is a compact subset of the unit square I^2. A point in the unit square is black if it belongs to R_∞; otherwise it is white.

In this article we consider various methods for generating the four types of patterns defined above. Typically finite patterns of different sizes are generated, which defines a multiresolution pattern. The multiresolution pattern can be interpreted in the limit either as an infinite pattern or as an infinite resolution pattern of the unit square in a manner made precise by Definition 2 below.

In the definition an interpretation of finite black-and-white patterns $P = (n, f)$ as compact subsets of the unit square I^2 is needed. Define the compact set

$$R_\infty(P) = \bigcup \left(\left[\frac{x}{n}, \frac{x+1}{n} \right] \times \left[\frac{y}{n}, \frac{y+1}{n} \right] \right),$$

where the union is over all $x, y \in \{0, 1, \ldots, n-1\}$ with $f(x, y) =$ 'black'. In other words, pattern P is scaled appropriately to fit inside the unit square. $R_\infty(P)$ is the infinite resolution interpretation of P.

Analogously, an interpretation of a finite pattern P as an infinite pattern is needed. This is obtained by placing P on the infinite square tessellation in a position specified by a given translation $(x, y) \in \mathbb{Z}^2$, and coloring all squares outside P using a fixed background color, usually 'white'. Formally, the infinite pattern $P_\infty(P) : \mathbb{Z} \to C$ is the infinite version of P with translation (x, y) if

$$P_\infty(P)(x + x', y + y') = \begin{cases} P(x', y') & \text{if } 0 \le x', y' \le n-1, \\ \text{'white'} & \text{otherwise.} \end{cases}$$

Definition 2. Let R be a multiresolution pattern defined by an infinite sequence $P_1, P_2, \ldots$ of finite patterns $P_i = (n_i, f_i)$.

(i) The multiresolution pattern R defines the infinite resolution pattern $R_\infty(R) \subseteq I^2$ if and only if $R_\infty(R)$ is the limit of the sequence $R_\infty(P_1)$, $R_\infty(P_2), \ldots$ of infinite resolution interpretations of finite patterns P_1, $P_2, \ldots$ in the standard Hausdorff topology on the compact subsets (see [1]). Intuitively, the patterns P_i are better and better approximations of the pattern $R_\infty(R)$. Note that not every multiresolution pattern defines an infinite resolution pattern—the limit is defined only if the sequence $R_\infty(P_1), R_\infty(P_2), \ldots$ converges.

(ii) Let $(x_1, y_1), (x_2, y_2), \ldots$ be an infinite sequence of elements of $\mathbb{Z}^2$ that describe how the finite patterns are to be placed on the infinite plane: The infinite version $P_\infty(P_i)$ of P_i is obtained by placing P_i with its lower-left corner in position (x_i, y_i). An infinite pattern $P_\infty(R) : \mathbb{Z}^2 \to C$ is defined by the multiresolution pattern R with translations (x_i, y_i) if for every $(x, y) \in \mathbb{Z}^2$ there exists a positive integer N such that $P_\infty(P_i)(x, y) = P_\infty(R)(x, y)$ for all $i \ge N$. In other words, $P_\infty(R)$ is the limit of the sequence $P_\infty(P_1), P_\infty(P_2), \ldots$ in the standard product topology on the set of infinite patterns. (The topology used is the infinite product of the discrete topologies on C; see for example [4].) The limit $P_\infty(R)$ is defined only if the sequence $P_\infty(P_1), P_\infty(P_2), \ldots$ converges.

If the sequence $(x_1, y_1), (x_2, y_2), \ldots$ of translations is not explicitly given it is assumed to be $(x_i, y_i) = (0, 0)$. In this case the finite patterns are placed with their lower-left corner in the origin of the plane. The infinite pattern they (possibly) define is contained in the upper-right quadrant of the plane; the other three quadrants have the background color.

The notation that we use should be understood as follows. $R_\infty(\cdot)$ is the infinite resolution pattern defined by its argument. If the argument is a finite pattern P, then $R_\infty(P)$ is obtained by properly scaling the pattern inside the unit square as described before Definition 2. If the argument is a multiresolution pattern R, then $R_\infty(R)$ is the limit as defined in Definition 2(i). Similarly, $P_\infty(\cdot)$ is the infinite pattern defined by its argument, which can be either a finite pattern P or a multiresolution pattern R.

In the following sections three devices for defining patterns are studied: cellular automata, finite automata (or equivalently, matrix substitution systems), and L-systems.

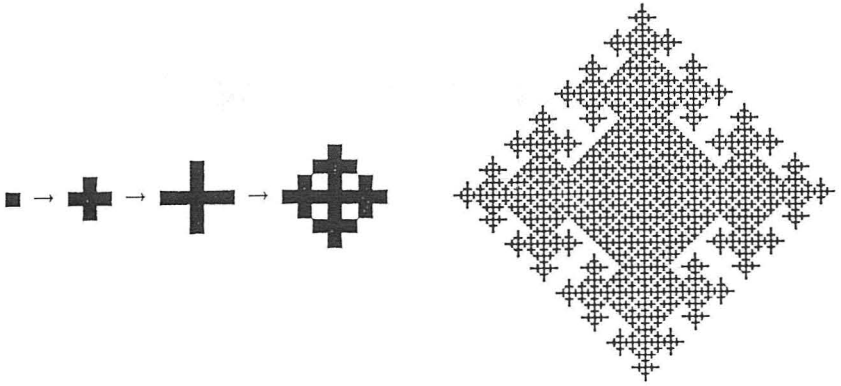
3. Two-dimensional cellular automata

Cellular automata are discrete dynamical systems used for computer simulations of various natural phenomena. Among the three models studied by this article, they are the lowest-level devices in the sense that they correspond to the basic physical laws covering pattern formation in nature.

Two-dimensional CAs operate on the infinite Euclidean plane divided into unit squares. The squares (referred to as cells) are indexed using integer coordinates. A finite *state set* S is fixed. At all times each cell is in one state of S. The cells alter their states synchronously at discrete time steps as specified by the *local transition rule* of the CA. The transition rule describes the new state of the cell as a function of the old states of some of the cell's neighbors. All cells use the same local rule. The *neighborhood* of the CA specifies which cells are considered the neighbors of a cell. Frequently used neighborhoods are the *Moore* neighborhood and the *von Neumann* neighborhood. The Moore neighborhood of a cell contains nine cells: the cell itself and the eight surrounding cells. In this case the local transition rule is a function from S^9 to S. In the von Neumann neighborhood the neighbors of a cell are the four closest cells—the cells above, below, and immediately to the right and left—as well as the cell itself.

A *configuration* of the CA is a mapping $\mathbb{Z}^2 \to S$ that specifies the states of all the cells. At each discrete time instance the present configuration of the CA is altered by applying the local rule simultaneously at all cells. When this process is repeated a sequence of configurations is obtained. This sequence describes the evolution of the CA. An evolution of infinite patterns is obtained if the states are interpreted as colors. For this purpose we define a coloring function $S \to C$ that gives the color of each state. Note that the coloring does not need to be one-to-one, which means that many states can be interpreted as the same color. The coloring function is extended in the obvious way to configurations, translating each configuration into an infinite pattern over the color set C.

One special state $q \in S$ is identified as the *quiescent state* of the CA. The quiescent state is usually assumed to satisfy the condition that each cell whose neighbors are all in the quiescent state remains quiescent on the next time instance. This guarantees that if initially only finitely many cells



(a) (b)

Figure 1: The evolution of Ulam's CA: (a) the beginning of the evolution, (b) the pattern produced after 60 iterations.

are non-quiescent, then this is true on subsequent configurations as well. A configuration with a finite number of non-quiescent cells is called *finite*. The coloring function is assumed to color the quiescent state q with the background color 'white'.

Now we are ready to define the infinite pattern generated by a CA A from a finite initial configuration c_0. Let $c_0, c_1, c_2, \ldots$ be the sequence of configurations obtained from c_0 by applying the CA rule repeatedly, and let $p_0, p_1, \ldots$ be the corresponding sequence of (infinite) patterns obtained from the configurations using the coloring function. The infinite pattern $p : \mathbb{Z}^2 \to C$ is generated by CA A from initial configuration c_0 if p is the limit of the sequence $p_0, p_1, \ldots$ in the product topology or, in other words, if for every position $(x, y) \in \mathbb{Z}^2$ there exists an integer N such that $p_i(x, y) = p(x, y)$ for all $i \geq N$.

Example 1. Consider the following very simple CA A introduced by Ulam. The CA has two states, 0 and 1. The state 0 is the quiescent state and is colored white, while state 1 is colored black. The CA A uses the von Neumann neighborhood. A cell in state 1 never changes its state. A cell in state 0 is changed into state 1 if and only if exactly one of its four closest neighbors is in state 1. The fact that state 1 never changes back to 0 guarantees that a limit exists for every initial configuration.

Figure 1 shows the first iteration steps of A starting from an initial configuration containing just one cell in state 1, as well as the pattern obtained after 60 iteration steps. Note the complex structure of the pattern even though the local rule of the CA is extremely simple. This "contradiction" between the simplicity of the local rule and the complexity of the pattern produced is typical of CAs.

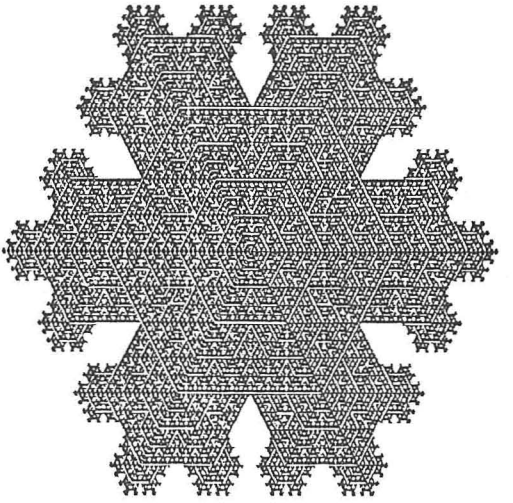
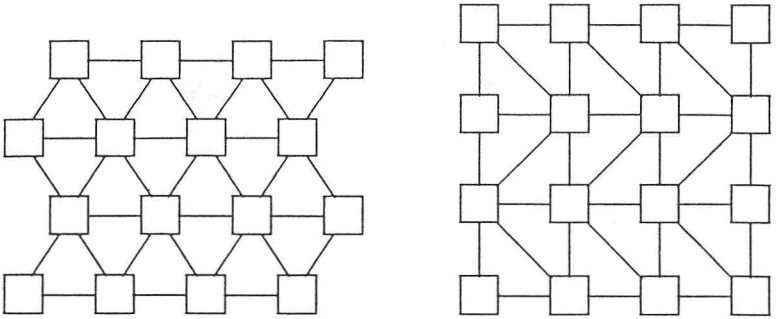


Figure 2: The snowflake produced by Ulam's CA on a hexagonal network.

The infinite pattern generated by A in the limit no longer has the crystal-like structure visible in Figure 1(b), which is due to the order in which the black states expand. The infinite pattern is a complex mixture of black and white cells, which, however, has the following simple arithmetic characterization: A cell in position (x, y) is white if and only if $x, y \neq 0$ and the binary representations of both x and y have the same number of 0s in the end. Note also how the black cells form a connected loop-free graph, or a tree.

Example 2. Consider the same simple local rule as in Example 1, but change the structure of the underlying network. A snowflake-like pattern of Figure 2 is obtained if the hexagonal network of Figure 3(a) is used. In fact, the hexagonal network can easily be simulated by a rectangular one if one new state is introduced: Instead of state 1 the CA has two black states, called 'even' and 'odd'. Intuitively, a black cell on an even row of the square tessellation is in state 'even', and on an odd row in state 'odd'. The neighborhood used is the Moore neighborhood. If the neighborhood of a cell in state '0' contains a black state, then the cell can deduce if it is on an even or odd row. Depending on this, the cell looks only at six of its neighbors, excluding cells in the left corners of the Moore neighborhood if it is on an even row and cells in the right corners if it is on an odd row. If exactly one of these six neighbors is black, the cell becomes black. With this simple trick, the network of Figure 3(b), isomorphic to the hexagonal network of Figure 3(a), is essentially used.

CAs define most naturally infinite patterns as described above. However, to compare CAs with the other pattern generation mechanisms introduced later in this article, let us describe how CAs define multiresolution patterns.



(a) (b)

Figure 3: (a) Hexagonal network. (b) Simulation of the hexagonal network on a rectangular one.

For this purpose, it is most convenient to use CAs that operate on finite square tessellations instead of the ideal infinite one. Let n be a positive integer, and consider a square of $n \times n$ cells. Initially all cells are in quiescent state q. The cells know if they have a neighbor that is outside the square, which means that the cells on the border can change during the first time step into a non-quiescent state. (The same effect is obtained by an infinite CA whose initial configuration contains a finite square island of quiescent states q in an infinite sea of states F that remain fixed throughout the computation.)

The finite CA A operates in the same way as the infinite one. The cells change their states synchronously by applying the local transition rule. The local rule must include the special cases of cells with neighbors outside the square. The CA is iterated until a configuration is obtained that does not change any more. If such a fixed point c_n is reached, the finite pattern p_n—obtained from c_n by using the coloring function—is the pattern of size n defined by A. If no fixed point is reached, A does not define any pattern of size n.

The sequence $p_{n_1}, p_{n_2}, \ldots$ of patterns of sizes $n_1 < n_2 < \cdots$ defined by A (where $n_1, n_2, \ldots$ includes all sizes at which A reaches a fixed point) is the multiresolution pattern $R(A)$ defined by A. The limits $R_\infty(A) = R_\infty(R(A))$ and $P_\infty(A) = P_\infty(R(A))$ given by Definition 2 are the infinite resolution pattern and infinite pattern, respectively, defined by CA A. To otain the infinite pattern $P_\infty(A)$ the finite squares are most conveniently placed on the infinite plane with their center in the origin. In this case it is not difficult to see that the exact same infinite patterns are generated in this way as can be generated by CAs on the infinite tessellation with the original, direct approach used in the beginning.

Example 3. Consider the following simple CA A with the von Neumann neighborhood and two states, representing black and white. A cell changes its state to the opposite color if and only if all its neighbors have the same

color as the cell itself. In addition, the cell in the lower-left corner of the finite tessellation (the cell whose left and lower neighbors are both outside the square) is turned black, regardless of the color of its other neighbors. Initially all cells are white. Clearly, in $2n-1$ steps the CA reaches the checkerboard configuration of alternating black and white squares that is no longer changed, where n is the size of the tessellation. The corresponding infinite pattern $P_\infty(A)$, obtained if the lower-left corner of each $n \times n$ checkerboard is placed in position $(-\lfloor \frac{n}{2} \rfloor, -\lfloor \frac{n}{2} \rfloor)$ on the infinite plane, is the infinite checkerboard. On the other hand, the infinite resolution pattern $R_\infty(A)$ is the completely black unit square. The infinite checkerboard is also obtained by the same local rule operating on the infinite tessellation if the initial configuration contains just one black cell.

4. Patterns generated by finite automata and iterative matrix substitution

Consider the tessellation of a square into $m^k \times m^k$ identical subsquares. We will assign strings of length k over the alphabet $\Sigma_m = \{0, 1, \ldots, m^2 - 1\}$ to the subsquares as unique addresses; that is, we give a bijection between the subsquares and $(\Sigma_m)^k$. Consider the address $a_1 a_2 \ldots a_k$, where $a_i \in \Sigma_m$. Let $r_i c_i$ be the two-digit m-ary representation of a_i, that is, $r_i, c_i \in \{0, 1, \ldots, m-1\}$. For example, for $m = 3$ and $a_i = 7$ we have $r_i = 2$ and $c_i = 1$. Denote by r and c the integers with m-ary representation $r_1 r_2 \ldots r_k$ and $c_1 c_2 \ldots c_k$, respectively. Then $a_1 a_2 \ldots a_k$ is assigned an address of the subsquare in the rth row and cth column, with the rows numbered $0, 1, \ldots, m-1$ bottom-up and the columns from left to right. The addresses of all the subsquares for $m = 2$, $k = 3$ and $m = 3$, $k = 2$ are shown in Figure 4.

Now we interpret a set of strings $S \subseteq (\Sigma_m)^k$ as a black-and-white pattern $P(S)$ of size $m^k \times m^k$. A subsquare of $P(S)$ is black if and only if its address is in S. Finally, we interpret a language $L \subseteq \Sigma_m^*$ as the multiresolution pattern $R(L)$ specified by the patterns $P(L_0), P(L_1), \ldots$ where $L_k = L \cap (\Sigma_m)^k$ for all $k \geq 0$. It is easy to see that the limit image $R_\infty(L) = R_\infty(R(L))$ (a compact subset of $[0, 1]^2$ as described in Definition 2) is exactly the image defined by L according to [2].

A great variety of multiresolution patterns can be specified by regular sets or more precisely by regular expressions or finite automata.

Example 4. Consider $\Sigma_2 = \{0, 1, 2, 3\}$ and $L = \Sigma_2^* \{0, 3\}$. Clearly, $R(L)$ is the multiresolution pattern such that $P(L_k)$ is the checkerboard of size $2^k \times 2^k$ (as in Example 3). For $L' = \{1, 2\}^* 0 \{1, 2\}^* 0 \Sigma_2^*$, the limit $R_\infty(L')$ is shown in Figure 5.

For any language L, using the operation of left quotient [6] we can zoom to the subsquare with address w of the image $R_\infty(L)$. The zoomed image is $R_\infty(w/L)$. For example, consider L', the diminishing triangles from Example 2. Zooming to any subsquare whose address u is a sequence of 1s and 2s gives us $u/L' = L'$; the image is partially self-similar. Zooming in the bottom-left

111	113	131	133	311	313	331	333
110	112	130	132	310	312	330	332
101	103	121	123	301	303	321	323
100	102	120	122	300	302	320	322
011	013	031	033	211	213	231	233
010	012	030	032	210	212	230	232
001	003	021	023	201	203	221	223
000	002	020	022	200	202	220	222

22	25	28	52	55	58	82	85	88
21	24	27	51	54	57	81	84	87
20	23	26	50	53	56	80	83	86
12	15	18	42	45	48	72	75	78
11	14	17	41	44	47	71	74	77
10	13	16	40	43	46	70	73	76
02	05	08	32	35	38	62	65	68
01	04	07	31	34	37	61	64	67
00	03	06	30	33	36	60	63	66

Figure 4: The squares addressed by strings in $(\Sigma_2)^3$ and $(\Sigma_3)^2$.

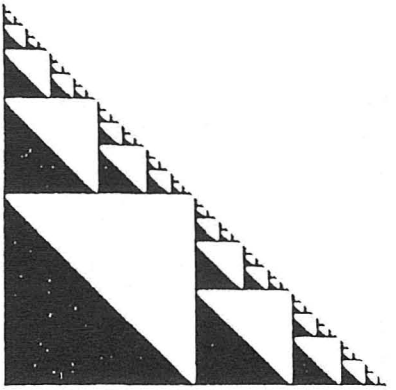


Figure 5: Diminishing triangles.

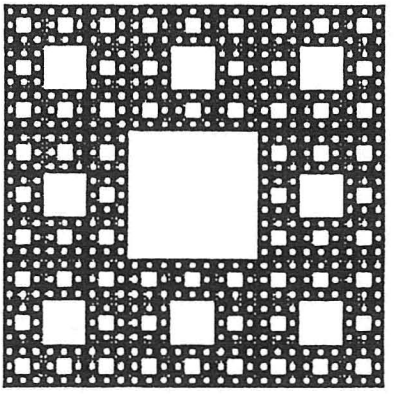


Figure 6: Cantor Carpet.

quadrant $0\backslash L' = \{1,2\}^*0\Sigma_2^*$ we get a regular set representing a triangle. For the upper-right quadrant we have $3\backslash L' = \emptyset$; the quadrant is completely white.

Example 5. Consider $\Sigma_3 = \{0,1,\ldots,8\}$ and $L'' = (\Sigma_3 - 4)^*$. The finite pattern $P(L_4'')$ (resolution $3^4 \times 3^4$) is shown in Figure 6. It is a good approximation of the ideal Cantor Carpet, which is represented by $R_\infty(L'')$. Note that for each $u \in (\Sigma_3 - 4)^*$ and each $v \in \Sigma_3^*$, $4v\backslash L'' = \emptyset$ and $u\backslash L'' = L''$, the image represented by $R_\infty(L'')$ is perfectly self-similar.

Example 6. As noted at the end of Example 1, the infinite pattern produced by Ulam's CA has a simple arithmetic characterization: A square in position (x,y) is white if and only if the binary expansions of x and y have the same number of trailing zeros. Thus the white squares in the upper-right quadrant of the plane are represented by the regular expression $\Sigma_2^*30^*$. The

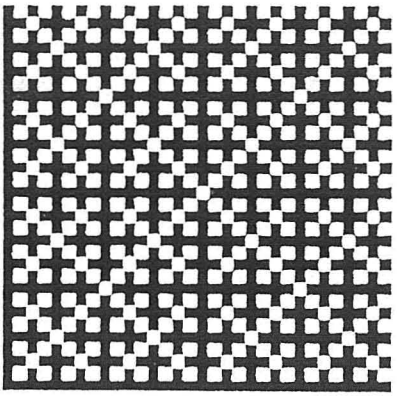


Figure 7: The pattern represented by $\Sigma_2^*\{1,2\}0^* \cup 0^*$ in resolution 32×32.

black squares are represented by its complement $\Sigma_2^*\{1,2\}0^* \cup 0^*$. The finite pattern at resolution $2^5 \times 2^5$ is shown in Figure 7.

Now we introduce matrix substitution systems (MSS), which are non-deterministic extensions of iterated matrix-valued homomorphisms from [9]. For an alphabet Δ we denote by $M_n(\Delta)$ the set of all $n \times n$ matrices with elements from Δ.

An MSS is a tuple $S = (n, \Delta, P, s, F)$ where

(i) $n \geq 1$, the size of the substitution matrices.

(ii) Δ is a finite set, the alphabet.

(iii) $P \subseteq \Delta \times M_n(\Delta)$ is the set of productions. A production $p = (a, C)$ is usually written $a \to C$. We write $\text{left}(p) = a$.

(iv) $s \in \Delta$, the initial symbol.

(v) $F \subseteq \Delta$, the symbols representing black squares.

S is called *deterministic* if for each $a \in \Delta$ there is exactly one production p in P with left side a, that is, with $\text{left}(p) = a$.

For $k \geq 1$, $A \in M_k(\Delta)$, and $B \in M_{nk}(\Delta)$ we write $A \Rightarrow B$ if B is the matrix obtained from matrix A by replacing each element, say a, by matrix $C \in M_n(\Delta)$ such that $a \to C \in P$. The reflexive and transitive closure of relations $\Rightarrow$ is denoted by $\Rightarrow^*$. We write $\Rightarrow^n$ for a derivation of exactly n steps.

Now, for $k \geq 1$, let $\pi_k = \{B \in M_{nk}(\Delta) \mid s \Rightarrow^k B\}$; that is, π_k is the set of all $n^k \times n^k$ matrices that can be derived from the initial symbol in k steps. We interpret π_k as the finite $n^k \times n^k$ pattern whose subsquare in the ith row and jth column is black if and only if $\bigcup_{B \in \pi_k} \{B_{i,j}\} \cap F \neq \emptyset$, that is, if the subsquare was "painted black" in at least one derivation of length k. The sequence $\pi_1, \pi_2, \ldots$ specifies the multiresolution pattern $R(S)$ defined by MSS S, which can be interpreted as an infinite resolution pattern

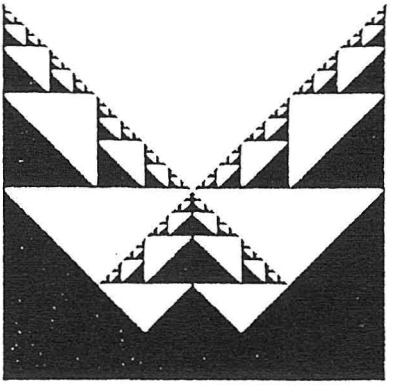


Figure 8: Double diminishing triangles.

$R_\infty(S) = R_\infty(R(S))$, or as an infinite size pattern $P_\infty(S) = P_\infty(R(S))$ as described in Definition 2.

Example 7. Let $S = (3, \{w, b\}, P, b, \{b\})$ where P contains productions

$$b \to \begin{vmatrix} b & b & b \\ b & w & b \\ b & b & b \end{vmatrix} \quad \text{and} \quad w \to \begin{vmatrix} w & w & w \\ w & w & w \\ w & w & w \end{vmatrix}.$$

S is a deterministic MSS that generates the Cantor Carpet of Figure 6.

Example 8. Let $S = (2, \Delta, P, v, \{1\})$ where $\Delta = \{v, d_1, d_2, t_1, t_2, 0, 1\}$ and P contains productions

$$v \to \begin{vmatrix} d_1 & 0 \\ t_1 & d_1 \end{vmatrix}, \quad v \to \begin{vmatrix} 0 & d_2 \\ d_2 & t_2 \end{vmatrix}, \quad d_1 \to \begin{vmatrix} d_1 & 0 \\ t_1 & d_1 \end{vmatrix}, \quad d_2 \to \begin{vmatrix} 0 & d_2 \\ d_2 & t_2 \end{vmatrix},$$
$$t_1 \to \begin{vmatrix} t_1 & 0 \\ 1 & t_1 \end{vmatrix}, \quad t_2 \to \begin{vmatrix} 0 & t_2 \\ t_2 & 1 \end{vmatrix}, \quad 0 \to \begin{vmatrix} 0 & 0 \\ 0 & 0 \end{vmatrix}, \quad 1 \to \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix},$$

The image $R_\infty(S)$ is shown in Figure 8.

Now we show that regular expressions and a (deterministic) MSS generate the same multiresolution patterns.

Theorem 1. *For every MSS S there effectively exists a deterministic MSS S' and a regular language L such that $R(S) = R(S') = R(L)$. For every regular language L there effectively exists a deterministic MSS S such that $R(S) = R(L)$.*

Proof.

1. Given $S = (n, \Delta, P, s, F)$ we construct a nondeterministic finite automaton $A = (\Delta, \Sigma_n, \delta, s, F)$ where the transition function δ is defined for every state $a \in \Delta$ and input symbol $j \in \Sigma_n$ as follows. State $b \in \Delta$ is in $\delta(a, j)$ if and only if there is a production $a \to C \in P$ and the element of C with address j is b. Clearly a string w is accepted by A if and only if the subsquare with addresses w in the $n^{|w|} \times n^{|w|}$ pattern generated by S is black.

2. Given a regular set $L \subseteq \Sigma_n^*$, there exists a complete deterministic finite automaton $A = (Q, \Sigma_n, \delta, q_0, F)$ such that $L(A) = L$. We construct MSS $S = (n, Q, P, q_0, F)$ where P is defined as follows. For each $q \in Q$, $q \to C \in P$ where $C \in M_n(Q)$ and the element of C with address j is $\delta(q, j)$. Clearly $R(L) = R(S)$.

3. Given S we obtain an equivalent S' using both previous steps of this proof. ∎

Without extending their generative power we can make MSS more convenient by allowing rotation or flipping of the substituted submatrices. For $A \in M_n(\Delta)$, $\rho(A)$ denotes the matrix obtained from A by the rotation 90° counterclockwise, and A^R denotes the left-right reversal (mirror image) of A. By composing 90° rotation and left-right reversal we can obtain any combination of 90°, 180°, and 270° rotations and any flipping. A production of an extended MSS $S = (n, \Delta, P, s, F)$ is of the form $a \to w$ where w is obtained by applying an arbitrary number of operations of rotation or reversal to a matrix $C \in M_n(\Delta)$. The extension of relation $\Rightarrow$ is obvious.

It is easy to see that by introducing additional symbols every extended MSS can be converted into an equivalent (deterministic) MSS. However, using an extended MSS might require fewer symbols as shown in the following examples.

Example 9. Let $S = (2, \Delta, P, v, \{1\})$ where $\Delta = \{v, d, t, 0, 1\}$ and P contains productions

$$v \to \begin{vmatrix} d & 0 \\ t & d \end{vmatrix}, \quad v \to \begin{vmatrix} 0 & d^R \\ d^R & t^R \end{vmatrix},$$
$$t \to \begin{vmatrix} t & 0 \\ 1 & t \end{vmatrix}, \quad 0 \to \begin{vmatrix} 0 & 0 \\ 0 & 0 \end{vmatrix}, \quad 1 \to \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix}.$$

Clearly $R_\infty(S)$ is the same as the one generated by the MSS of Example 8 and shown in Figure 8.

Example 10. In [9] a deterministic MSS that generates the Peano's space-filling curve is shown. It has 17 symbols. We will generate Peano's curve

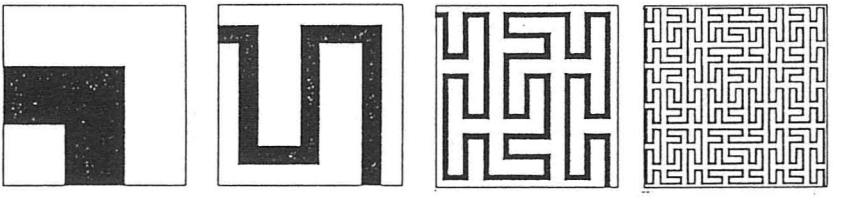


Figure 9: Peano's space-filling curve.

with the following extended MSS with only four symbols. $S = (3, \{a, b, 0, 1\}, P, a, \{1\})$ where P contains productions

$$a \to \begin{vmatrix} a & \rho(a) & a \\ b^R & b & b^R \\ \rho^2(a) & \rho^3(a) & b \end{vmatrix}, \quad a \to \begin{vmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix},$$

$$b \to \begin{vmatrix} b & \rho(a) & a \\ b^R & b & b^R \\ \rho^2(a) & \rho^3(a) & b \end{vmatrix}, \quad b \to \begin{vmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix}.$$

See Figure 9 for the first steps of the extended MSS S.

The following theorem was proved in [5].

Theorem 2. *For every regular language L there effectively exists a cellular automaton A such that $R(A) = R(L)$. Moreover, CA A generates the multiresolution pattern $R(L)$ in linear time; that is, there exists a constant c $(= 2)$ such that the finite pattern $P(L_k)$ of size $m^k \times m^k$ is generated by A in $O(c \cdot m^k)$ time steps, for every $k > 0$.*

According to Theorem 2, cellular automata are efficient parallel devices for simulating finite automata or iterative matrix substitutions. Note that any sequential algorithm for drawing a pattern of size $m^k \times m^k$ requires at least $(m^k)^2$ operations.

5. L-systems

The third class of pattern-generating mechanisms considered here are L-systems. It is well known that L-systems are convenient tools for simulating biological growth. We use only the most elementary L-systems, called DOL-systems, which consist of simple, context-free, rewriting rules of strings. The strings generated are interpreted as patterns using turtle geometry [7].

Formally, a DOL-system is a triplet $D = (\Sigma, h, w)$ where Σ is a finite alphabet, $h : \Sigma^* \to \Sigma^*$ is a morphism, and $w \in \Sigma^+$ is a non-empty word called an *axiom*. The morphism h is a collection of *rewriting rules* $a \to u$, where $a \in \Sigma$ is a letter and $u \in \Sigma^*$ is a word. For every letter a of Σ there is exactly one rewriting rule with a on the left side—the system is

deterministic. The word u on the right-hand side of the production is the homomorphic image of a (under h).

For an arbitrary word $v \in \Sigma^*$ we write $v \Rightarrow v'$ if v' is the word obtained from v by replacing each letter by its homomorphic image. The DOL-system D generates the infinite sequence $w_0, w_1, \ldots$ of words, where $w_0 = w$ and, for every $k \geq 1$, $w_{k-1} \Rightarrow w_k$.

Next we explain how a word w_k is interpreted as an image using turtle geometry. Our approach is similar to the one introduced by Prusinkiewicz [7]; the only difference is that because we compare L-systems with cellular automata and finite automata that define patterns composed of colored squares, the turtle in our interpretation draws squares, not lines. A *turtle* is a simple drawing device that moves on the infinite plane divided into unit squares. The *state* of the turtle is a triplet (x, y, α), where (x, y) are integer coordinates of the position of the turtle, and $\alpha \in \{0°, 90°, 180°, 270°\}$ is the angle indicating the direction in which the turtle is heading. Initially the state of the turtle is $(0, 0, 90°)$, meaning that the turtle is in the origin and facing up. Then the string w_k is scanned from left to right, interpreting letters as commands to the turtle in the order they are encountered.

The alphabet Σ is assumed to contain the following special symbols: F, f, $+$, $-$, [, and]. In addition there may be an arbitrary number of other symbols. The rewriting rules for $+$, $-$, [, and] are restricted: each of them is rewritten to itself. Symbols F and f, as well as any auxiliary symbols, may have unrestricted rewriting. As the string w_k is scanned the six special symbols are interpreted as commands to the turtle as follows:

- F Paint the square under the turtle black, and advance one unit in the present direction.
- f Advance one unit on the plane in the present direction (without painting anything).
- $+$ Turn left (counterclockwise) 90°.
- $-$ Turn right (clockwise) 90°.
- [Push the current state of the turtle onto a stack.
-] Pop a state from the top of the stack and make it the current state of the turtle.

All other letters of the alphabet do not affect the turtle; they are just used to direct the evolution of the L-system.

Note that the only change to the original turtle interpretation by Prusinkiewicz is in the interpretation of F. Also the directions in our approach are restricted to straight angles.

Example 11. Let $D_1 = (\{A, F, f, +, -, [,]\}, h, A)$ where the rewriting rules for A, F, and f are

$$A \to A[+fA][-fA]FA,$$

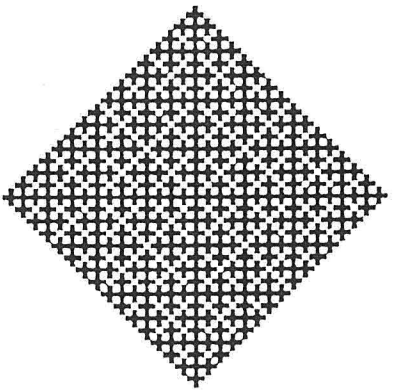


Figure 10: The sixth iterate of the DOL-system simulating Ulam's CA.

$$F \to F,$$
$$f \to f.$$

The first strings generated by the DOL-system D_1 are

$$\begin{aligned} w_0 &= A, \\ w_1 &= A[+fA][-fA]FA, \\ w_2 &= A[+fA][-fA]FA[+fA[+fA][-fA]FA][-fA[+fA][-fA]FA] \\ &\quad FA[+fA][-fA]FA. \end{aligned}$$

The graphical interpretation of the string w_k is the same pattern that was produced by Ulam's CA in $2^{k-1} - 1$ steps for every $k > 0$ (see Example 1). The interpretation of w_6 is depicted in Figure 10.

Most conveniently the sequence $p_0, p_1, \ldots$ of patterns obtained with the turtle interpretation from the sequence $w_0, w_1, \ldots$ of strings is understood as a sequence of infinite patterns. The limit in the product topology, if it exists, is the infinite pattern generated by the DOL-system D. To translate the sequence $p_0, p_1, \ldots$ into a multiresolution pattern, the borders of the finite patterns need to be specified. Let $S = (x_0, y_0, n_0), (x_1, y_1, n_1), \ldots$ be an infinite sequence of triples of integers, where $0 < n_0 < n_1 < \cdots$. The multiresolution pattern $R(D)$ defined by DOL-system D together with sequence S is given by the finite patterns $(n_0, f_0), (n_1, f_1), \ldots$ where, for every $k > 0$, $f_k(x, y) = p_k(x + x_k, y + y_k)$ for $x, y \in \{0, 1, \ldots, n_k - 1\}$. In other words, the finite pattern (n_k, f_k) is obtained from the infinite one p_k by taking only the finite portion inside the square of size $n_k \times n_k$ whose lower-left corner is in position (x_k, y_k).

Theorem 3. *For every regular language L there effectively exists a DOL-system D such that $R(D) = R(L)$.*

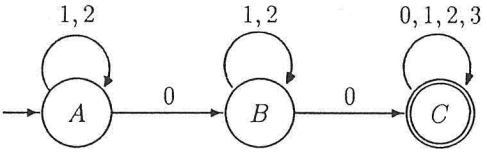


Figure 11: A deterministic finite automaton for the language $L' = \{1,2\} * 0\{1,2\} * 0\Sigma_2^*$.

Proof. Given regular language $L \subseteq \Sigma_m^*$, there exists a complete deterministic finite automaton $A = (Q, \Sigma_m, \delta, q_0, E)$ such that $L(A) = L$. We construct DOL-system $D = (Q \cup \{F, f, +, -, [,]\}, h, q_0')$ where the rewriting rules in h are defined below. For each $i \in \Sigma_m$ denote $w_i = -f^{c_i} + f^{r_i}$, where $r_i c_i$ is the two-digit m-ary representation of i. Clearly w_i is the command string that makes the turtle move from its initial state to the subsquare with address i. Also, denote $q' = qF$ if $q \in E$ and $q' = q$ if $q \in Q \backslash E$. (The string q' commands the turtle to paint the square black if q is a final state.)

The rewriting rule for symbol $q \in Q$ is

$$q \to [w_0\, \delta(q,0)'][w_1\, \delta(q,1)'] \ldots [w_{m^2-1}\, \delta(q, m^2-1)'].$$

The rewriting rules for the special symbols F and f are

$$F \to \epsilon,$$
$$f \to f^m,$$

where ϵ is the empty word.

The axiom of the DOL-system D is q_0', that is, $q_0 F$ if $q_0 \in E$ and q_0 if $q_0 \notin E$. If the sequence of patterns generated by D is interpreted as a multiresolution pattern $R(D)$ using triple $(0, 0, m^k)$ for cutting the finite portion from the infinite plane on the kth iteration (see discussion before Theorem 3), it is easy to get convinced that $R(D) = R(L)$. ∎

Note that the number of auxiliary symbols in the DOL-system constructed in the proof of Theorem 3 is the same as the number of states in the complete deterministic automaton recognizing the language. The construction can be modified in a straightforward manner to work directly for an arbitrary (nondeterministic) finite automaton, thus reducing the number of auxiliary symbols needed in the DOL-system.

Example 12. Consider the finite automaton shown in Figure 11 recognizing the language L' of Example 4. The construction in the proof of Theorem 3 produces the equivalent DOL-system $(\{A, B, C, F, f, +, -, [,]\}, h, A)$ where the rewriting rules are as follows:

$$A \to [B][fA][-f + A],$$
$$B \to [CF][fB][-f + B],$$
$$C \to [CF][fCF][-f + CF][-f + fCF],$$
$$F \to \epsilon,$$
$$f \to ff.$$

(The construction was simplified by not including the expressions corresponding to nonexisting transitions in the automaton of Figure 11—the automaton is not complete.)

6. Conclusions

We have compared the generative powers of three pattern generation mechanisms: cellular automata, finite automata, and L-systems. Cellular automata have the highest generative power among the three models. This is due to the fact that CAs are computationally universal. Consequently, any recursive infinite pattern can be generated using CAs. On the other hand, to find the local rule for generating a particular pattern appearing in nature can be difficult: to simulate the natural process using a CA one typically simulates the basic physical laws. In this sense CAs correspond to the lowest, most basic level of simulation.

L-systems have the second highest generative power. Introduced by Lindenmayer in 1968 as models for the development of certain organisms, they provide a natural mechanism for simulating the growth of biological structures. L-systems operate on the higher, biological level of simulation.

Finite automata have the most restricted generative power among the three mechanisms. On the other hand, given a pattern, if a finite automaton exists that generates a given pattern, it is easy to find. Finite automata do not construct patterns through simulation of natural processes; they directly express the self-similarities in the patterns.

Acknowledgments

This research was supported by NSF Grant No. CCR-9202396.

References

[1] M. F. Barnsley, *Fractals Everywhere* (San Diego: Academic Press, 1988).

[2] K. Culik II and S. Dube, "Rational and Affine Expressions for Image Description," *Discrete Applied Mathematics*, **41** (1993) 85–120.

[3] K. Culik II and J. Kari, "Image Compression Using Weighted Finite Automata," *Computer and Graphics* (to appear).

[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, "How to Fire Almost Any Pattern on a Cellular Automata," *Proceedings of the NATO Workshop on Cellular Automata and Cooperative Systems*, Les Houches, France, June–July 1992 (to appear).

[6] John E. Hopcroft and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Reading, MA: Addison-Wesley, 1979).

[7] P. Prusinkiewicz, "Graphical Applications of L-systems," *Proceedings of Graphic Interface 1986—Vision Interface*, (1986) 247–253.

[8] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants* (New York: Springer Verlag, 1990).

[9] J. Shallit and J. Stolfi, "Two Methods for Generating Fractals," *Computer and Graphics*, **13** (1989) 185–191.