International Journal of Bifurcation and Chaos, Vol. 12, No. 12 (2002) 2655–2766

A NONLINEAR DYNAMICS PERSPECTIVE OF WOLFRAM'S NEW KIND OF SCIENCE. PART I: THRESHOLD OF COMPLEXITY

LEON O. CHUA
Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley,
Berkeley, California, CA 94720, USA

SOOK YOON
Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley,
Berkeley, California, CA 94720, USA

RADU DOGARU
Department of Electrical Engineering and Computer Sciences,
Polytechnic University of Bucharest, Bucharest, Romania

Received August 28, 2002; Revised November 10, 2002

This *tutorial* provides a *nonlinear dynamics* perspective to Wolfram's monumental work on *A New Kind of Science*. By mapping a *Boolean local Rule*, or *truth table*, onto the *point attractors* of a specially tailored nonlinear *dynamical system*, we show how some of Wolfram's *empirical* observations can be justified on firm ground. The advantage of this new approach for studying Cellular Automata phenomena is that it is based on concepts from nonlinear dynamics and *attractors* where many fuzzy concepts introduced by Wolfram via brute force observations can be defined and justified via mathematical analysis. The main result of *Part I* is the introduction of a fundamental concept called *linear separability* and a *complexity index* κ for each *local Rule* which characterizes the intrinsic *geometrical structure* of an induced "Boolean cube" in three-dimensional Euclidean space. In particular, Wolfram's seductive idea of a "*threshold of complexity*" is identified with the class of local Rules having a complexity index equal to 2.

Keywords: Cellular Automata; CNN; Cellular Neural Networks; Cellular Nonlinear Networks; local Rule; A New Kind of Science; Linearly Separable Rules; complexity; threshold of complexity; S. Wolfram.

1. Introduction 2656
2. Cellular Automata is a Special Case of CNN 2657
3. Every Local Rule is a Cube with Eight Colored vertices 2658
4. Every Local Rule is a Code for Attractors of a Dynamical System 2660
   4.1. Dynamical System for Rule 110 2732
   4.2. There are Eight Attractors for Each Local Rule 2733
5. Every Local Rule has a Unique Complexity Index 2737
   5.1 Geometrical Interpretation of Projection σ and Discriminant $w(\sigma)$ 2737

5.2 Geometrical Interpretation of Transition Points of Discriminant Function $w(\sigma)$. 2738
5.3 Geometrical Structure of Local Rules 2740
5.4 A Local Rule with Three Separation Planes 2741
5.5 Linearly Separable Rules 2749
5.6 Complexity Index 2750
5.7 Every Local Rule is a Member of an Equivalence Class 2756
5.8 Making Non-Separable from Separable Rules 2764
5.9 Index 2 is the Threshold of Complexity 2765

1. Introduction

The objective of this *tutorial* is to provide a *nonlinear dynamics* perspective to Stephen's Wolfram's beautifully articulated masterpiece on *A New Kind of Science* [Wolfram, 2002], which is based almost entirely on *empirical* observations from computer simulations. In particular, we will develop a *geometrical* approach for defining an integer characterization of *all* Boolean functions arising from one-dimensional Cellular Automata with nearest neighbors (our theory, however, is valid for any dimension and with any neighborhood size). This integer, called the *complexity index* κ, is an *intrinsic* measure of the *structural complexity* of every local *Rule*. We will show that the complexity index provides a rigorous definition for Wolfram's insightful but fuzzy concept on "Threshold of complexity", a seductive idea without a definition!

Our object of study in this paper is a *ring* of *coupled cells* C_i, $i = 0, 1, 2, \ldots, N$, as shown in Fig. 1(a). For maximum generality, each cell C_i is assumed to be a dynamical system, shown in Fig. 1(b), with an intrinsic state x_i, an *output* y_i, and three *inputs* u_{i-1}, u_i, and u_{i+1}, where u_{i-1} denotes the input coming from the *left* neighboring cell C_{i-1}, u_i denotes the "self" input to cell C_i, and u_{i+1} denotes the input coming from the *right* neighboring cell C_{i+1}. Each cell evolves in accordance with its prescribed dynamics, and has its own time scale. When coupled together, the resulting system would evolve in a way that is consistent with its own "rule" as well as the "rule of *interaction*" imposed by the "coupling laws". For the purpose of this paper, we assume each *input* is a constant *integer* $u_i \in \{-1, 1\}$, and the *output* $y_i(t)$ converges to a constant $y_i \in \{-1, 1\}$ from a zero initial condition $x_i(0) = 0$ (a standing assumption in this paper). In the context of Cellular Automata, we ignore the fact that it takes a finite amount of time for any dynamical system to converge to an *attractor* and idealize the situation by assuming each attractor is reached instantaneously. Under this assumption and in view of the *binary* nature of both the *input* and the *output*, our dynamical system is equivalent to a nonlinear *map* which can be uniquely described by a *truth table* of three input variables (u_{i-1}, u_i, u_{i+1}), called a *local Rule* in [Chua, 1998] and [Wolfram, 2002]. Our choice of $\{-1, 1\}$, and not the conventional symbols $\{0, 1\}$ as our *binary* signals, is not merely cosmetic but absolutely crucial in this paper because we will map our truth table onto a *dynamical system* where the state x_i and output y_i evolve in real time via a carefully designed scalar *ordinary differential equation* which is carefully designed so that after the solution $x_i(t)$ (with zero initial state $x_i(0) = 0$) reaches a steady state, the *output* $y_i(t)$ (which is defined via an *output equation* $y_i = y(x_i)$) tends to either 1 or -1. In other words, we will use the attractors of the dynamical system to encode a binary truth table.

Aside from the cell's intrinsic time scale, which is of no concern in Cellular Automata, we will introduce an external clocking mechanism which *resets* the input u_i of each cell C_i at the end of each clock cycle by feeding back the steady state (i.e. attractor) output $y_i \in \{-1, 1\}$ as an updated input $u_i \in \{-1, 1\}$ for the next iteration. The resulting system is called a one-Dimensional Cellular Automata with a periodic boundary condition. Notice that although cellular automata is concerned only with the ring's evolutions over *discrete* times, any system or computer used to simulate cellular automata is always a continuous time system with a very small but *non-zero* time scale. Even the personal computer which Stephen Wolfram uses to create his spectacular collection of evolved patterns

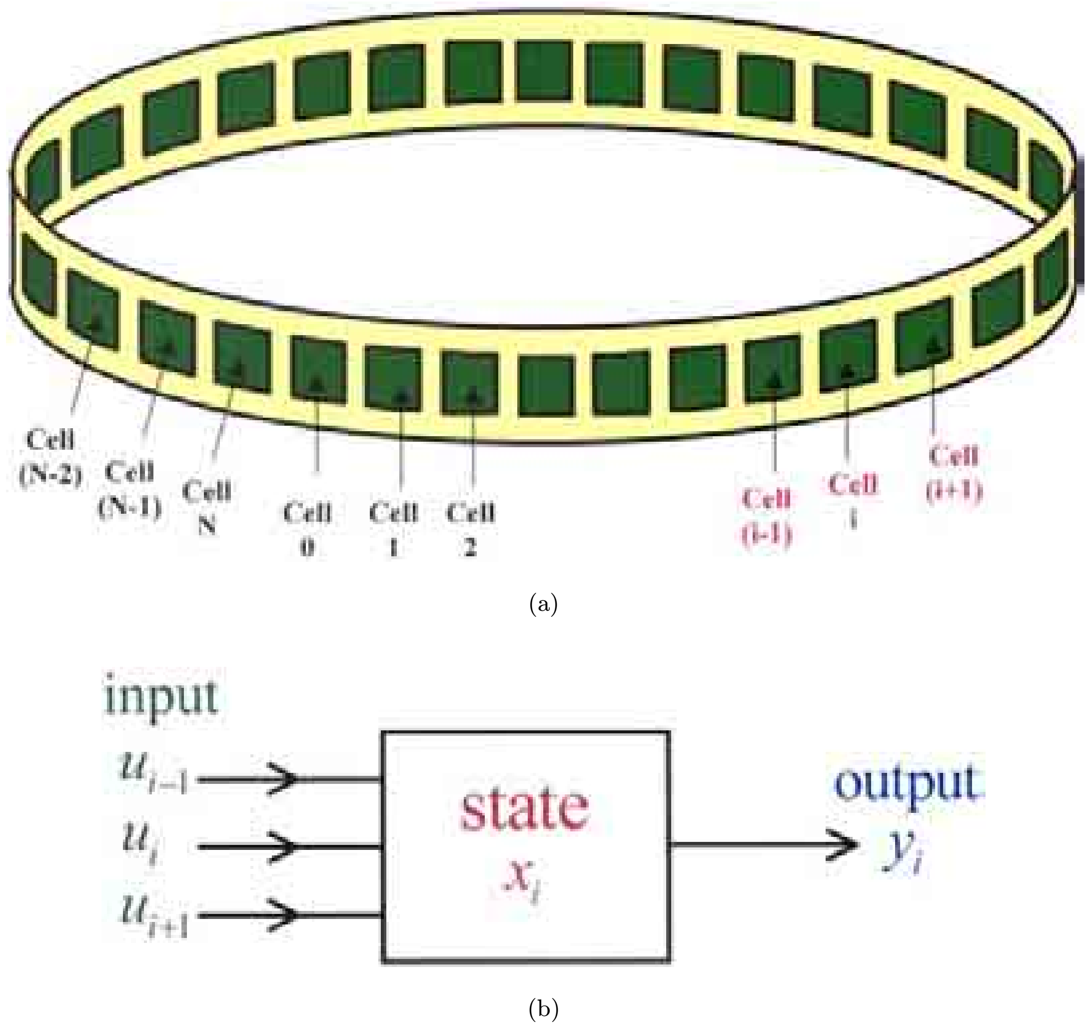


Fig. 1. (a) A one-dimensional Cellular Automata (CA) made of $(N+1)$ identical cells with a periodic boundary condition. Each cell "i" is coupled only to its left neighbor cell $(i-1)$ and right neighbor cell $(i+1)$. (b) Each cell "i" has a state variable $x_i(t)$, an output variable $y_i(t)$ and three constant binary inputs u_{i-1}, u_i, and u_{i+1}.

are made of devices called transistors, and each cellular automata iteration involves the physical evolution of several million transistors, each having its own intrinsic dynamics. These transistors evolve in accordance with a very large system of nonlinear differential equations governing the entire internal computer circuit and return the desired output after converging to their respective attractors in a *non-zero* amount of time, which translates into the computer's processing speed.

What we wish to emphasize here is that even in discrete systems like cellular automata, there are two different time scales involved. The first applies to the *local Rule* while the second applies to the *global* patterns of evolution. To understand the complex dynamics of global patterns, it is *necessary* to examine *both* mechanisms.

This paper (Part I) is concerned only with the mathematical characterization of local Rules. By unfolding a "lifeless" truth table into an appropriate nonlinear dynamical system, we can exploit the theory of nonlinear differential equations [Shilnikov *et al.*, 1998; Shilnikov *et al.*, 2001] to arrive at a phenomena based on sound mathematical theory, and not on empirical observations.

2. Cellular Automata is a Special Case of CNN

CNN is an acronym for either Cellular Neural Network when used in the context of brain science, or Cellular Nonlinear Networks when used in the context of coupled dynamical systems [Chua *et al.*, 1995; Chua, 1998; Chua & Roska, 2002]. A CNN is defined by two mathematical constructs:

1. A spatially discrete collection of nonlinear *dynamical systems* called *cells*,[1] where *information* can be encrypted into each cell via three independent variables called *input*, *threshold* and *initial state*.
2. A *coupling law* relating one or more relevant variables, such as *state*, *output*, etc., of each cell C_{ij} to all neighbor cells C_{kl} located within a prescribed *sphere of influence*[2] $s_{ij}(r)$ of radius r, centered at C_{ij}.

In the special case where the CNN consists of a homogeneous array, and where its cells have no inputs, no thresholds, and no outputs, and where the sphere of influence extends only to the nearest neighbors, the CNN reduces to the familiar concept of a nonlinear *lattice*.

From a technological perspective, CNN represents currently the only practical method for fabricating a cell array of meaningful size for image processing applications. Because of its *local connectivity* ($r = 1$), it is currently possible to cram more than 4 million CMOS transistors into a 128×128 CNN chip on 1 square centimeter area of silicon and dissipating less than 4 watts of power [Liñan *et al.*, 2002].[3] This chip, called a *CNN Universal Chip*, as well as several other competing chips, can be *programmed* via a user-friendly language so that instead of implementing only one evolution law per chip, an entire sequence of evolution laws can be programmed and executed all on *the same chip* just like a personal computer. In this case, however, we have an enormously more powerful and orders of magnitude faster computer because every CNN cell in the array is processing information *simultaneously*, a truly *parallel* computer on a chip! For many mission critical applications, such as tracking a missile in flight, an earlier 64×64 CNN chip has already outperformed a conventional supercomputer in terms of processing power.

We will prove in Sec. 4 that for each one-dimensional cellular automata with nearest neighbors and *any* prescribed local Boolean function of three binary variables (u_{i-1}, u_i, u_{i+1}), we can design a CNN cell defined by a *scalar* nonlinear differential equation whose corresponding output tends to an *attractor* which codes the desired local rule. Moreover, all of these cells have the same "structural" form in the sense that a *single* scalar nonlinear differential equation can be tuned to yield a correct binary output consistent with any prescribed local rule by simply choosing eight real numbers. Such a task can be easily implemented on a CNN either by straightforward programming, or by using a designer's CNN cell and executed in a few nanoseconds with current technology. We will illustrate both avenues in Sec. 4. Furthermore, it will follow from our analysis in Sec. 4 that our theory is independent of the size of the sphere of influence, as well as, on the spatial dimension of the CNN. In other words, we have the following fundamental result[4]:

Theorem 1. *Every binary cellular automata of any spatial dimension is a special case of a CNN with the same neighborhood size.*

3. Every Local Rule is a Cube with Eight Colored Vertices

A Boolean function is usually described in computer science or informatics by a *truth table* where each binary variable is represented symbolically by either a "0" or a "1". In this paper, it is *absolutely essential* that we use "−1" and "1" instead of "0" and "1" because these variables, except in a few *strictly* Boolean settings, must be interpreted as *real* numbers in all subsequent mathematical analysis and calculations, such as solving differential equations, which are all based on the *real* number system. Hence, the truth table for a Boolean function of three binary variables u_{i-1}, u_i, and u_{i+1} will be depicted as in the upper part of Fig. 2. The only exception to this assumption is in the output y_i of cell "C_i" where we may revert back to "0" and "1" whenever it does not enter into

[1]The cells need not be identical and are usually arranged uniformly on a two- or three-dimensional orientable manifold in space, e.g. rectangular, hexagonal, toroidal, spherical arrays, etc. The variables may assume *continuous* values or a finite number of discrete symbols. The *dynamical system* may be specified by an *evolution law* or *algorithm*, such as a *differential equation*, a *difference equation*, an *iterative map*, a *semigroup*, etc.

[2]For example, $r = 1$ for all nearest neighbors and $r = 2$ for all nearest and next nearest neighbors.

[3]In contrast, because of its full connectivity where each cell is coupled to every other cell, the famous Hopfield network has remained only as a useful conceptual tool.

[4]We conjecture that Theorem 1 holds not only for binary cellular automata, but also for any *finite* number of states.

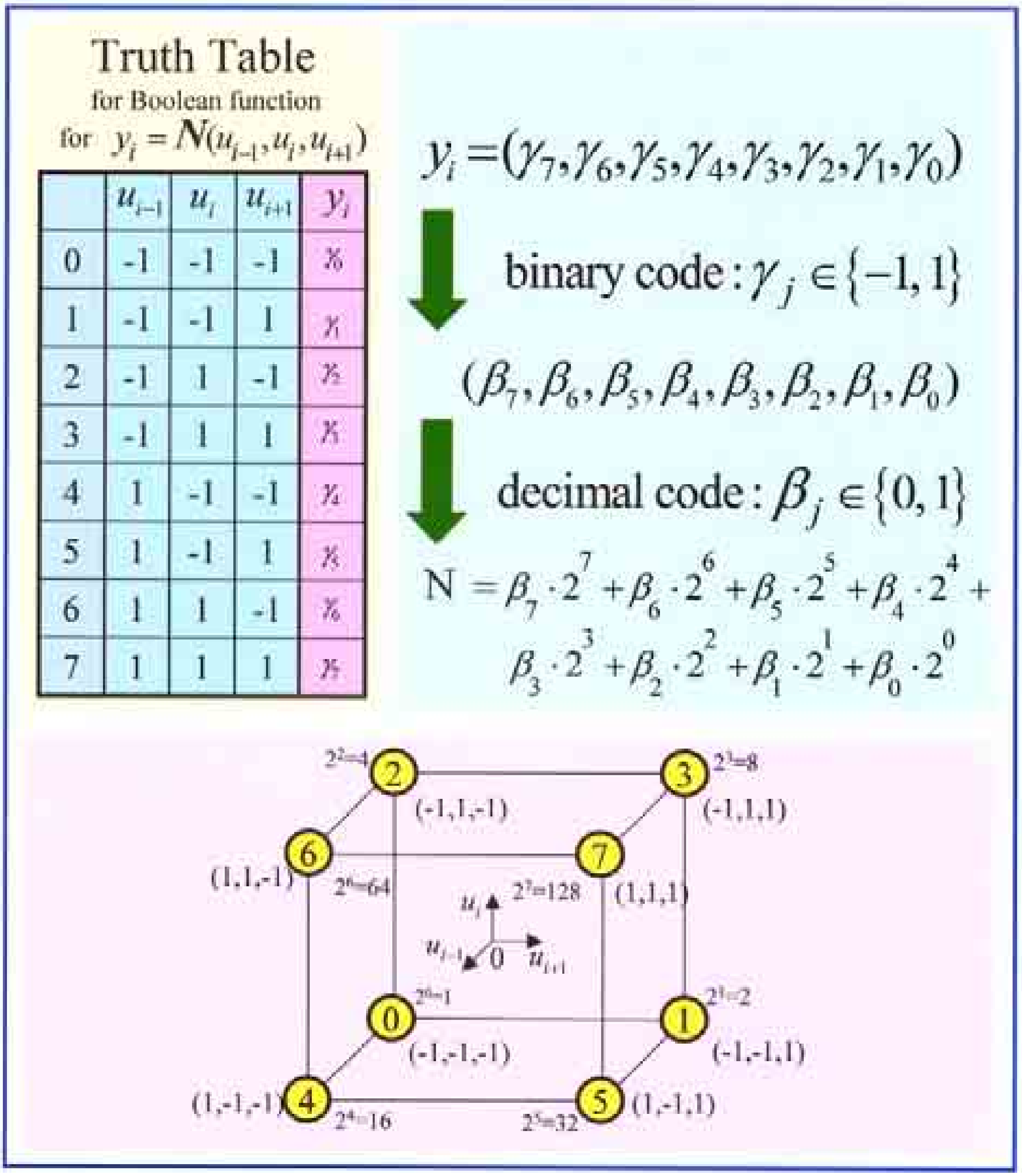


Fig. 2. Each Boolean function of three binary variables can be uniquely represented by a Boolean cube with colored vertices. The center of the cube is located at the origin of the three-dimensional (u_{i-1}, u_i, u_{i+1})-space. The coordinates of each vertex ⓚ correspond to row k of the truth table. The number 2^k shown next to vertex ⓚ is its decimal equivalent.

any arithmetic or algebraic calculations. For example, it is more convenient to decode the output $y_i = (\gamma_7, \gamma_6, \gamma_5, \gamma_4, \gamma_3, \gamma_2, \gamma_1, \gamma_0)$ in *decimal* system by recasting it into its equivalent binary form $y_i = (\beta_7, \beta_6, \beta_5, \beta_4, \beta_3, \beta_2, \beta_1, \beta_0)$ where β_j is either a "0" or "1" so that the corresponding decimal number is simply the integer $N = \beta_7 \bullet 2^7 + \beta_6 \bullet 2^6 + \beta_5 \bullet 2^5 + \beta_4 \bullet 2^4 + \beta_3 \bullet 2^3 + \beta_2 \bullet 2^2 + \beta_1 \bullet 2^1 + \beta_0 \bullet 2^0$, as shown in the upper part of Fig. 2. Since there are $2^{2^3} = 256$ distinct combinations of this eight-bit word, there are exactly 256 distinct Boolean functions, each one identified uniquely by an integer N, where $N = 0, 1, 2, \ldots, 255$. It is important to observe that the output y_i specifies either a *Boolean rule* (when coded in "−1" and "1") or its *identification number* (when coded in "0" or "1"). This "dual" role assumed by y_i follows the convention adopted in [Chua, 1998; Wolfram, 2002].

Since the three binary variables u_{u-1}, u_i, and u_{i+1} in a one-dimensional Cellular Automata are coded in terms of *real* numbers "−1" and "1", we can identify each input (u_{i-1}, u_i, u_{i+1}) as a *vertex* of a *cube* (of length 2 on each side) centered at the *origin*. It is also extremely convenient to refer to each of these eight vertices by an integer $0, 1, 2, \ldots, 7$, by reverting back to its corresponding three-bit binary word. For example, the binary word associated with the vertex located at $(-1, 1, 1)$ is 011, which decodes into the integer 3. In other words, we can identify uniquely each vertex of the Boolean cube by an integer n, where n ranges from 0 to 7. We will henceforth adopt this identification scheme, as depicted in the lower part of Fig. 2, where each vertex number is enclosed by a circle.

Observe next that if we paint each vertex ⓝ *red* when y_i is "1", or *blue*, when y_i is "−1" in row "n" of the truth table, then the resulting "Boolean cube" contains exactly the same information as the truth table. This simple equivalent description of a Boolean function of three binary variables represents not only a very compact description, it also turns out to be crucially important in Sec. 5 where the cube's spatial geometry will be fully exploited to arrive at a unique characterization of the *structural complexity* of a Boolean function.

Each of the 256 Boolean cubes is listed in Table 1 along with its identification number, henceforth called its rule number.[5] Note that each *rule number* is printed either in *red*, or in *blue*, which codes for a *Linearly Separable rule*, and a *Linearly Non-Separable rule*, respectively. The significance of these two classes of Boolean rules will be revealed in Sec. 5.

Given any Boolean cube from Table 1, we can easily identify its *rule number* by simply adding the decimal number 2^k for each integer k associated with a *red* vertex. For example, the decimal numbers associated with the five red vertices ①, ②, ③, ⑤ and ⑥ (for rule 110) is equal to $2^1 + 2^2 + 2^3 + 2^5 + 2^6 = 110$, which can be trivially read off the decimal numbers shown next to each vertex in Fig. 2, viz. $2+4+8+32+64 = 110$, as expected.

4. Every Local Rule is a Code for Attractors of a Dynamical System

Our main result of this paper is to provide a *constructive* and *explicit* proof that every *Boolean function*, or local *rule*, N from Table 1 can be mapped into a nonlinear *dynamical system* whose *attractors encode* precisely the associated truth table N, $N = 0, 1, 2, \ldots, 255$. In particular, the dynamical system can be chosen to be a *scalar* ordinary *differential equation* of the form.[6]

$$\begin{aligned} \dot{x}_i &= g(x_i) + w(u_{i-1}, u_i, u_{i+1}) \\ x_i(0) &= 0 \end{aligned} \tag{1}$$

where

$$g(x_i) \triangleq -x_i + |x_i + 1| - |x_i - 1| \tag{2}$$

henceforth called the *driving-point function*,[7] and $w(u_{i-1}, u_i, u_{i+1})$ is a scalar nonlinear function of three *real* variables u_{i-1}, u_i, and u_{i+1} for each local rule N, $N = 0, 1, 2, \ldots, 255$. In particular, $w(u_{i-1}, u_i, u_{i+1})$ can be chosen [Dogaru & Chua, 1999], to be a *composite* function $w(\sigma)$ of a single

[5]Table 1 is a reordered version of Fig. 57 from [Chua, 1998], where the binary rule number y_i of the truth table in Fig. 2 was decoded with γ_7 as the least significant digit.

[6]There are many possible choices of nonlinear "basis function" for $g(x_i)$ and $w(u_{i-1}, u_i, u_{i+1})$, such as polynomials. We have chosen the absolute value function $f(x) = |x|$ as our nonlinear "basis function" in this paper not only because the resulting equation can be expressed in an optimally compact form which fits the limited space provided in Table 2, but also because it allows us to derive the solution of Eq. (1) in an explicit form. Moreover, it is much easier to build *chip* for implementing Eq. (1) with absolute-value functions via current microelectronics technology.

[7]Equation (1) is solved explicitly in [Chua, 1969] for any *piecewise-linear* driving-point function. The terminology "driving point" comes from nonlinear circuit theory and need not concern readers of this paper. Many other driving-point functions which produce the same local rule N can be chosen; for example, we can choose $g(x_i) = x_i - x_i^3$, which looks simpler mathematically, but much more difficult to implement on a chip.

Table 1. List of 256 Boolean Function "Cubes" defining all Boolean functions of three binary variables.

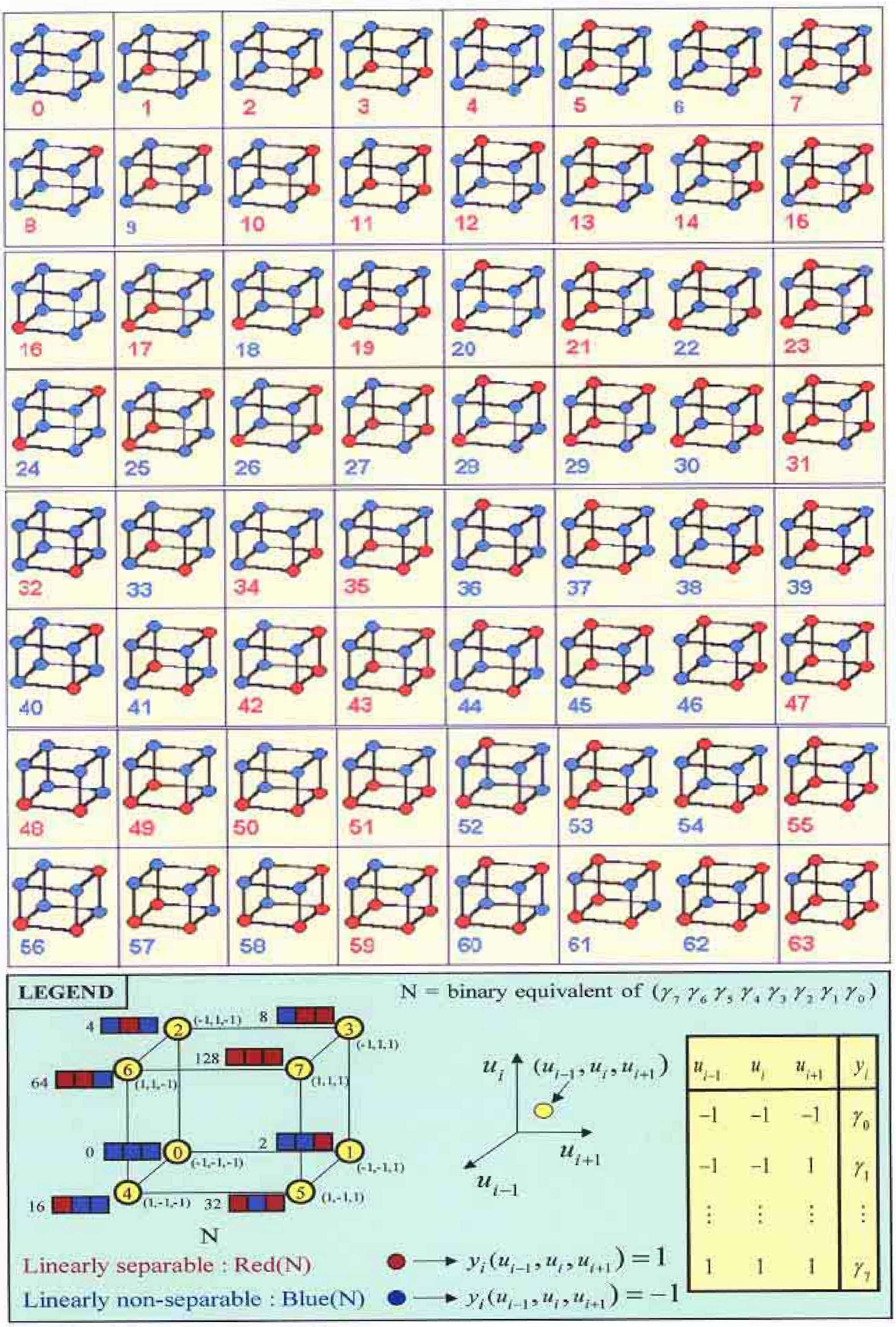
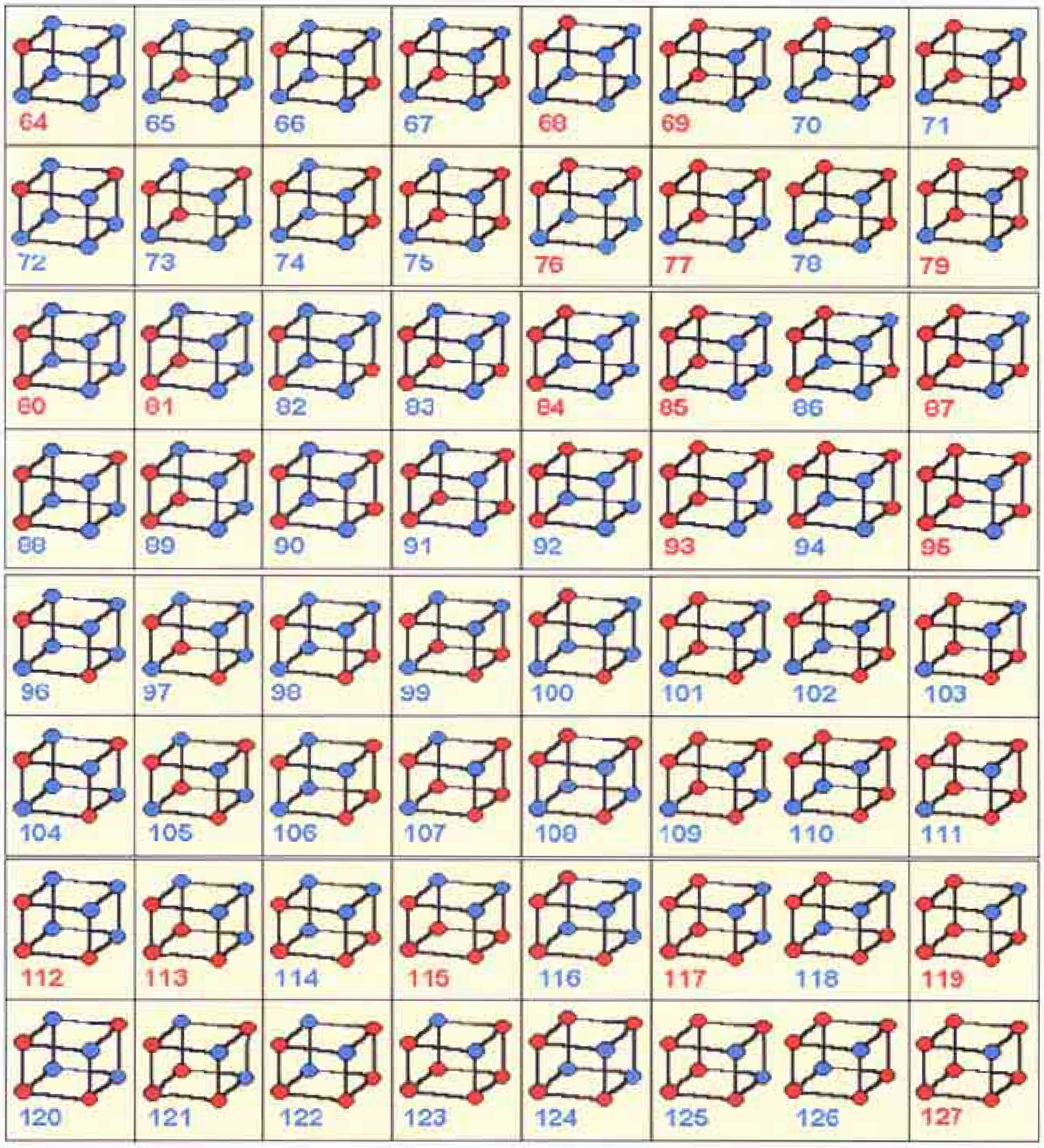
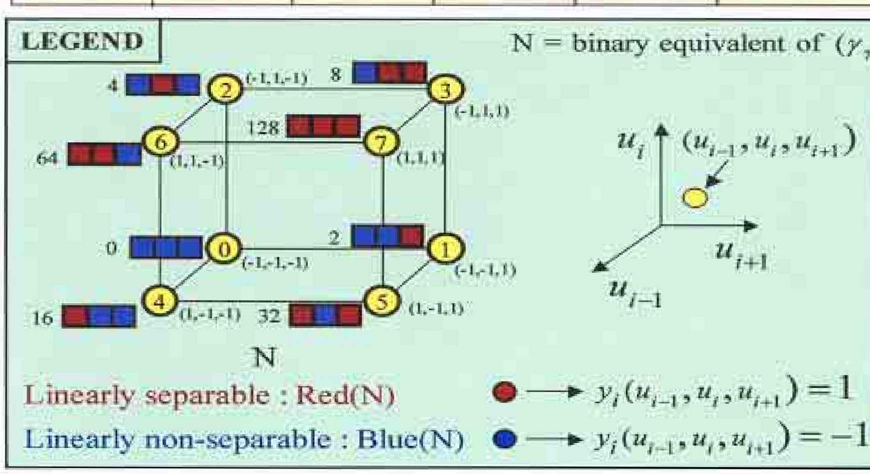


Table 1. (*Continued*)





u_{i-1}	u_i	u_{i+1}	y_i
−1	−1	−1	γ_0
−1	−1	1	γ_1
⋮	⋮	⋮	⋮
1	1	1	γ_7

Table 1. (*Continued*)

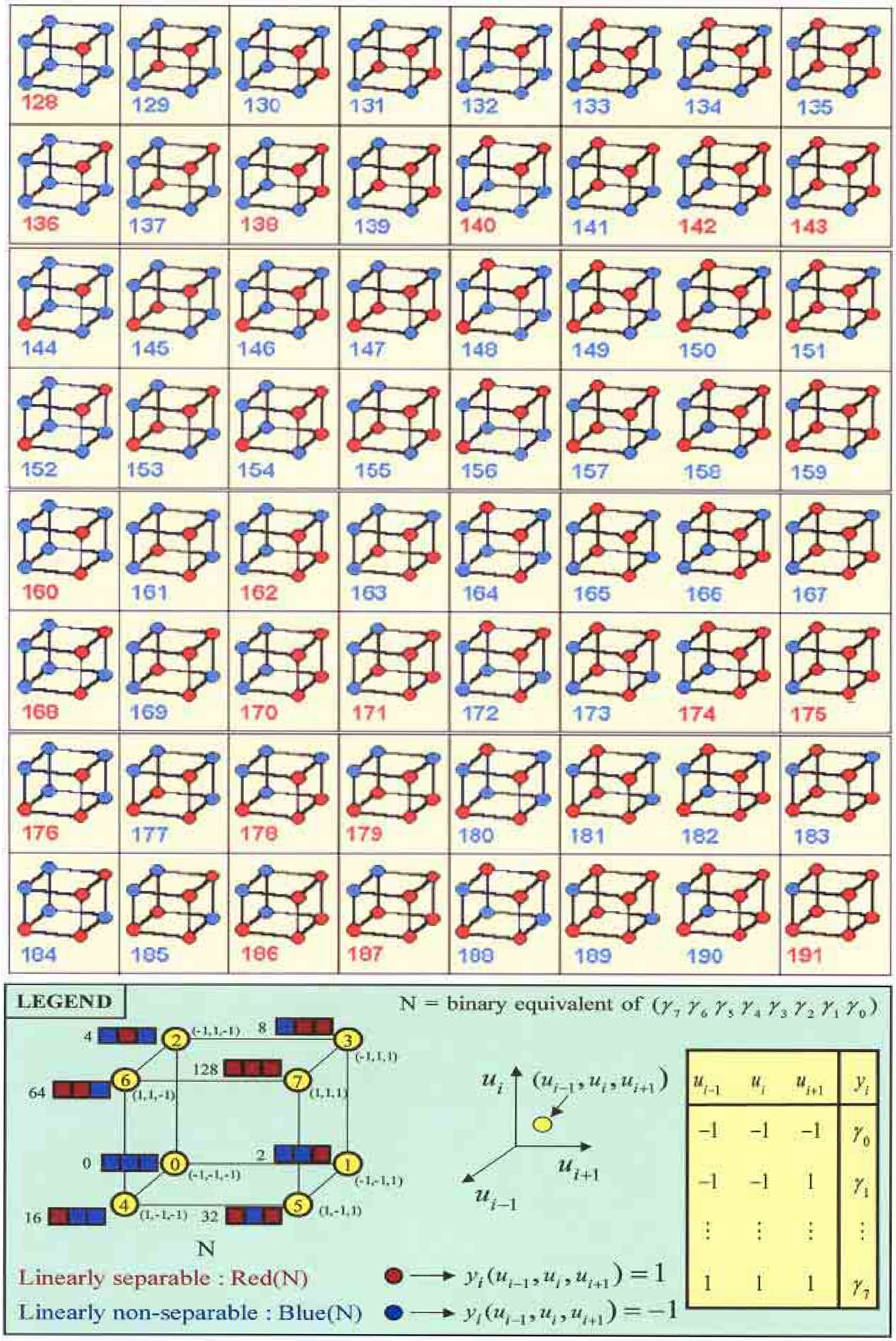
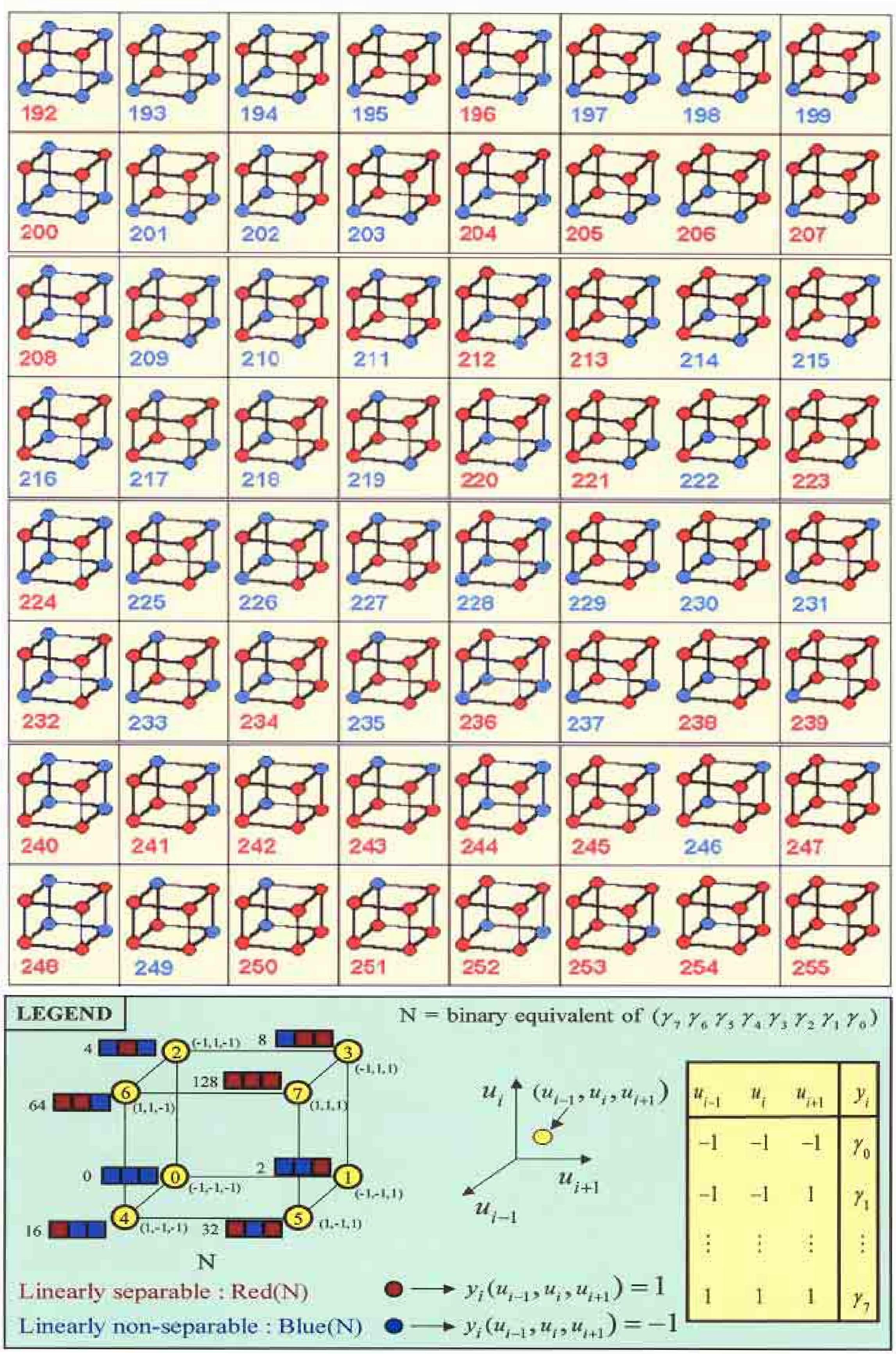


Table 1. (*Continued*)



variable

$$\sigma \triangleq b_1 u_{i-1} + b_2 u_i + b_3 u_{i+1} \triangleq \mathbf{b}^T \mathbf{u} \quad (3)$$

henceforth called a *projection* in this paper, where

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \text{and} \quad \mathbf{u} = \begin{bmatrix} u_{i-1} \\ u_i \\ u_{i+1} \end{bmatrix} \quad (4)$$

are called an *orientation vector* and *input vector*, respectively, and where

$$w(\sigma) \triangleq \{z_2 \pm |[z_1 \pm |z_0 + \sigma|]|\} \quad (5)$$

Depending on the context where it is most meaningful, the composite function $w(\sigma)$ is called a *discriminant function*, or an *offset level* [Chua, 1998] in this paper. Observe that even though the discriminant $w(\sigma)$ is a function of three input variables u_{i-1}, u_i, and u_{i+1} in view of Eq. (3), it is a *scalar* function of only *one* variable σ. This rather special mathematical structure of $w(\sigma)$ is the single most important property which makes it such a delightfully simple task to map each local rule onto a nonlinear *dynamical system* and painlessly perceive its trajectories converging into various attractors which can then be coded in a truth table in a one-to-one manner!

The same *discriminant function* $w(\sigma)$ is used to define the appropriate differential equation (1) for generating the truth table of *all* 256 Boolean cubes listed in Table 1. Each local rule corresponds to a particular set of six real numbers $\{z_2, z_1, z_0; b_1, b_2, b_3\}$, and two integers ± 1. All together only eight parameters are needed to uniquely specify the differential equation (1) associated with each local rule N, $N = 0, 1, 2, \ldots, 255$. Since eight bits are needed to specify each local rule, or the colors of eight vertices are needed to specify each Boolean cube, the discriminant function is optimal in the information-theoretic sense that it calls for only the *minimum number* of information needed to uniquely specify a Boolean function of three binary variables.

We will prove below that once the parameters defining a particular local rule N from Table 1 are specified, then for any one of the eight inputs (u_{i-1}, u_i, u_{i+1}) listed in the *truth table* in Fig. 2, the solution $x_i(t)$ of the scalar differential equation (1) will either increase monotonically from the *initial state* $x_i = 0$ toward a *positive equilibrium* value $\overline{x}_i(n) \geq 1$, henceforth denoted by *attractor* $Q_+(n)$, or decrease monotonically towards a *negative equilibrium state* $\overline{x}_i(n) \leq -1$, henceforth denoted by *attractor* $Q_-(n)$, when the input (u_{i-1}, u_i, u_{i+1}) in Eq. (1) is chosen from the coordinates of vertex ⓝ of the associated Boolean cube in Fig. 2; or equivalently, from row "n" of the truth table in Fig. 2, for $n = 0, 1, 2, \ldots, 7$. Observe that if we paint vertex ⓝ *red* whenever its equilibrium value $\overline{x}_i(n) \geq 1$, and *blue* whenever $\overline{x}_i(n) \leq -1$, then the color of all eight vertices for the associated Boolean cube will be uniquely specified by the equilibrium solutions of the eight associated differential equations. If we simulate Eq. (1) with a *chip*, the equilibrium state will be attained in only a *few nanoseconds* (10^{-9} seconds), which is practically instantaneous for many real-world applications.

In short, once the parameters associated with a particular *local rule* from Table 1 are specified, the corresponding *truth table* or *Boolean cube*, will be uniquely generated by the scalar differential equation (1) alone. Note, however, that the equilibrium value of $\overline{x}_i(n)$ is not equal in general to ± 1 and is thereby *not* a binary number although we have managed to assign a color correctly to each vertex, under the implicit "understanding" that vertex ⓝ will be coded *red* if $\overline{x}_i(n) \geq 1$, or *blue* if $\overline{x}_i(n) \leq -1$.

To avoid having to make such an ad hoc assumption, we will formally identify each local rule in Table 1 by a *dynamical system* defined as follows:

State Equation

$$\begin{aligned} \dot{x}_i &= f(x_i; u_{i-1}, u_i, u_{i+1}) \\ x_i(0) &= 0 \end{aligned} \quad (6)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|) \quad (7)$$

Observe from Eq. (7) that $y_i = +1$ when $x_i \geq 1$, and $y_i = -1$ when $x_i \leq -1$, thereby making it unnecessary to introduce the harmless though *ad hoc* assumption.

The dynamical systems for generating all 256 local rules in Table 1 are compiled in Table 2 for future reference. In each case, the scalar differential equation (1) is obtained by substituting the expression $y(x_i)$ from the *output equation* in place of the

Table 2. Dynamical system for generating all 256 local rules listed in *Table 1*. The differential equation is obtained by substituting the *output equation* for y_i in the *state equation*. The initial condition is $x_i(0) = 0$.

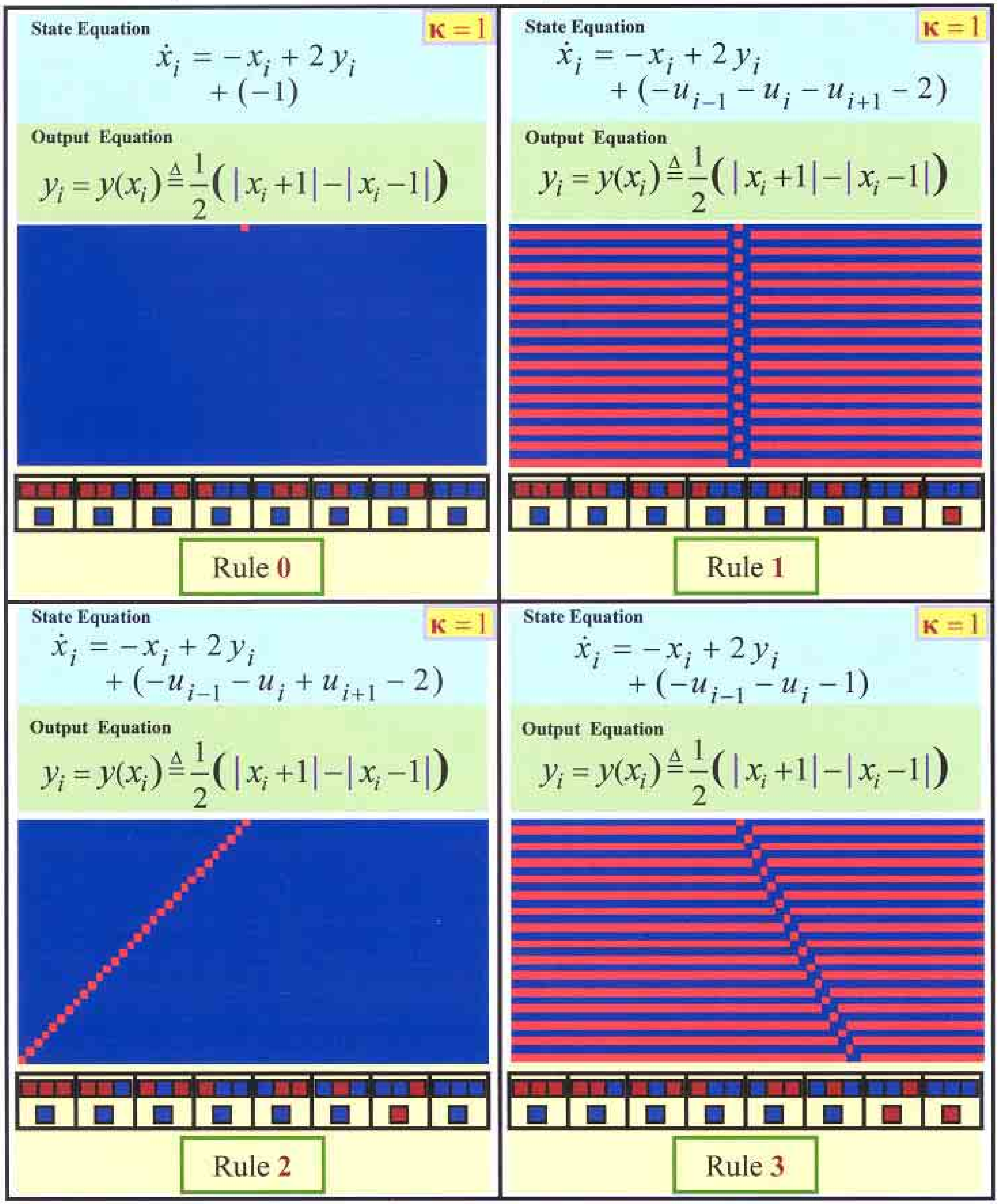


Table 2. (*Continued*)

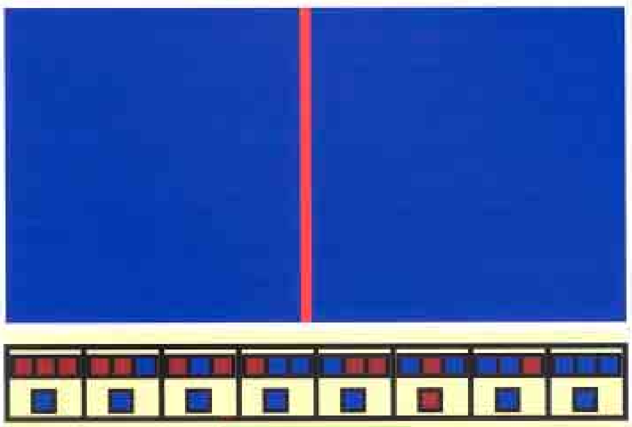
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} + u_i - u_{i+1} - 2)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}\left(|x_i + 1| - |x_i - 1|\right)$$



Rule **4**

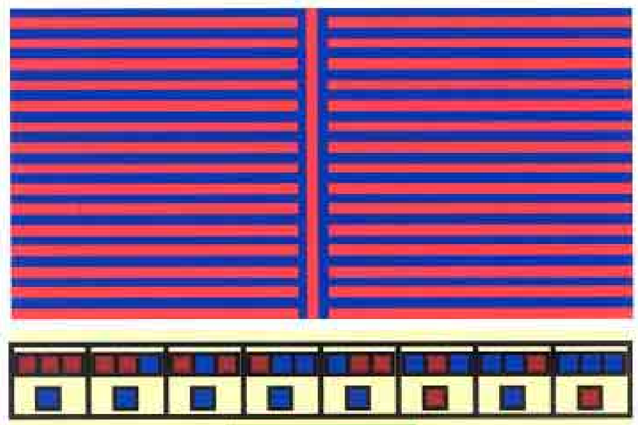
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} - u_{i+1} - 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}\left(|x_i + 1| - |x_i - 1|\right)$$



Rule **5**

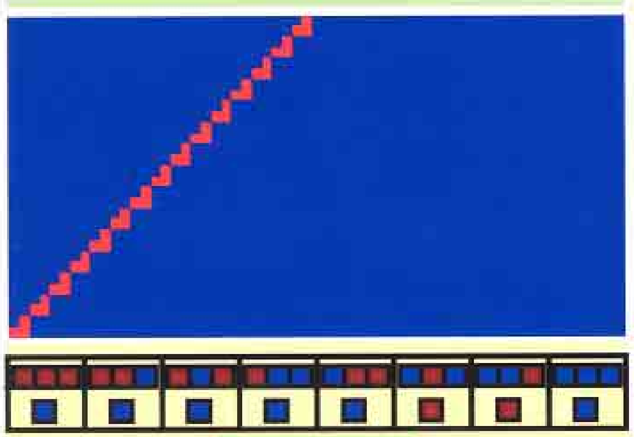
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [1 - |(2u_{i-1} - u_i - u_{i+1} + 2)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}\left(|x_i + 1| - |x_i - 1|\right)$$



Rule **6**

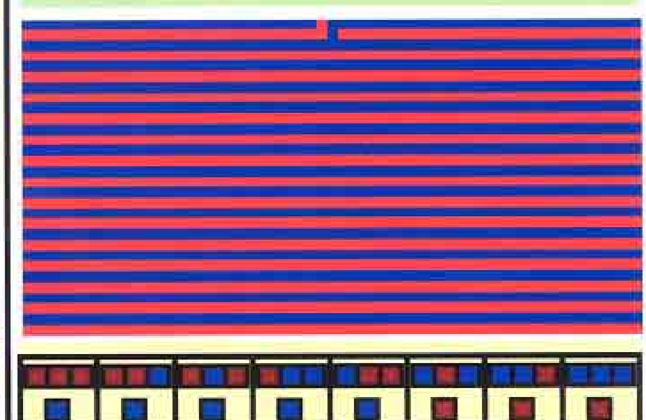
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-2u_{i-1} - u_i - u_{i+1} - 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}\left(|x_i + 1| - |x_i - 1|\right)$$



Rule **7**

Table 2. (*Continued*)

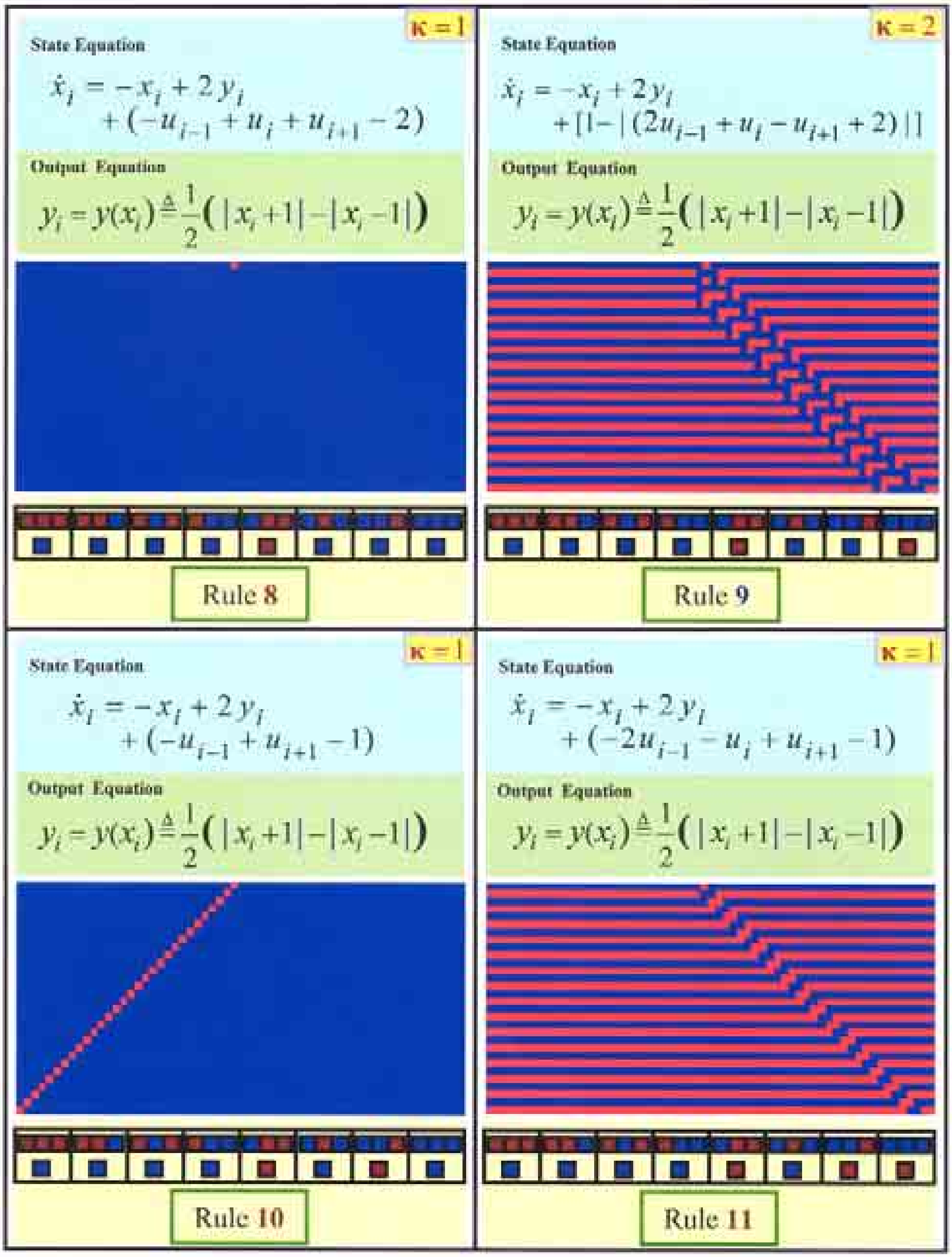


Table 2. (*Continued*)

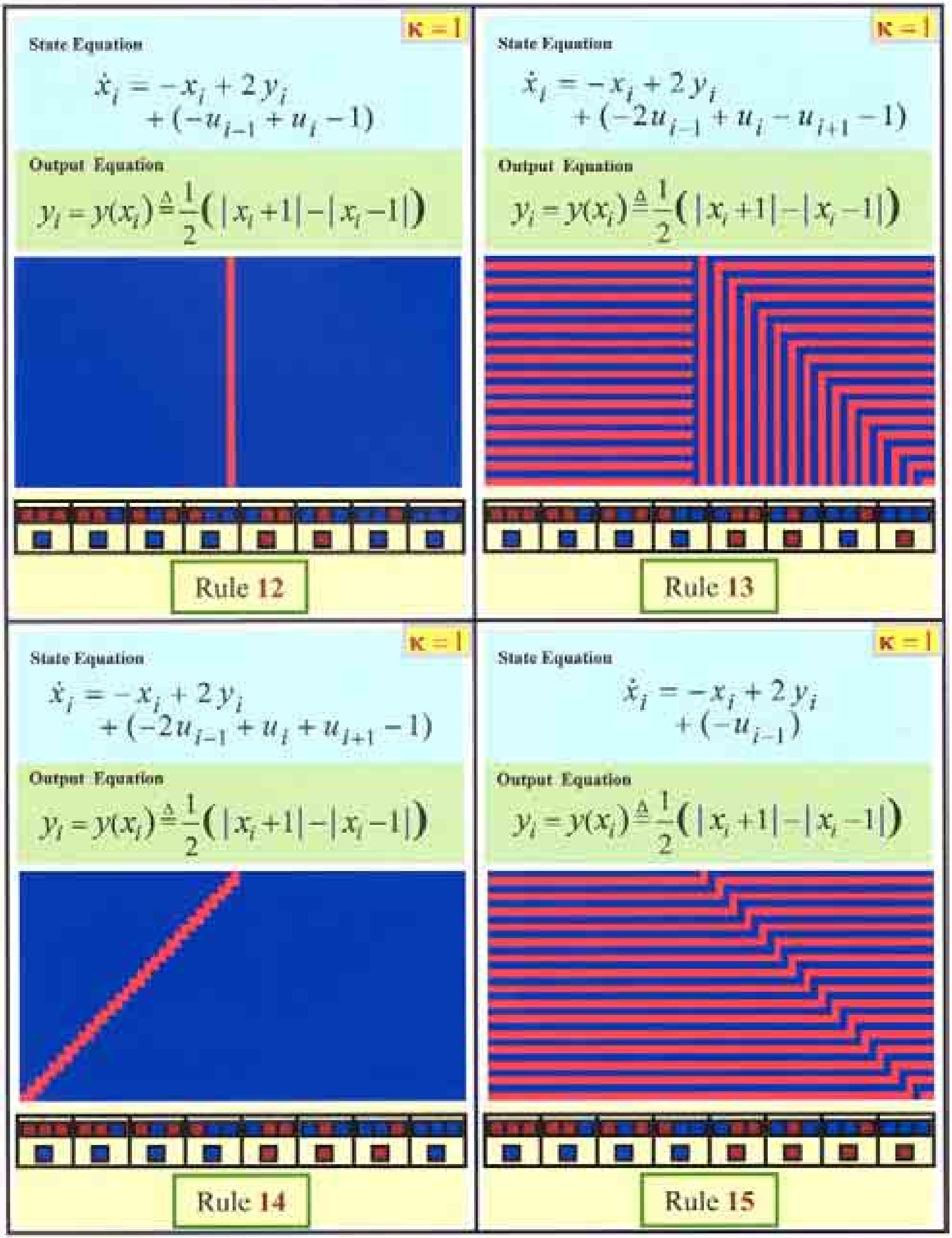


Table 2. (*Continued*)

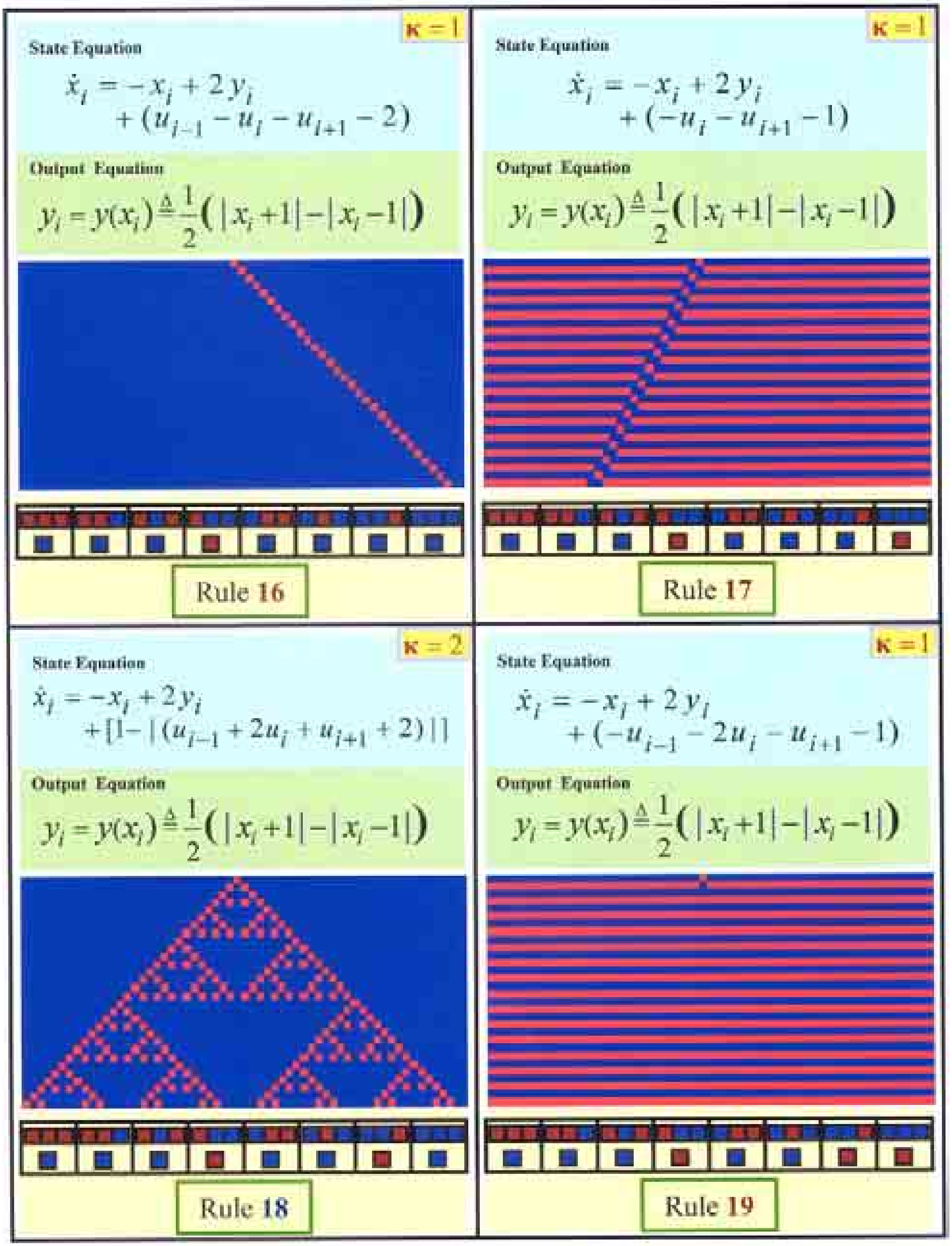


Table 2. (*Continued*)

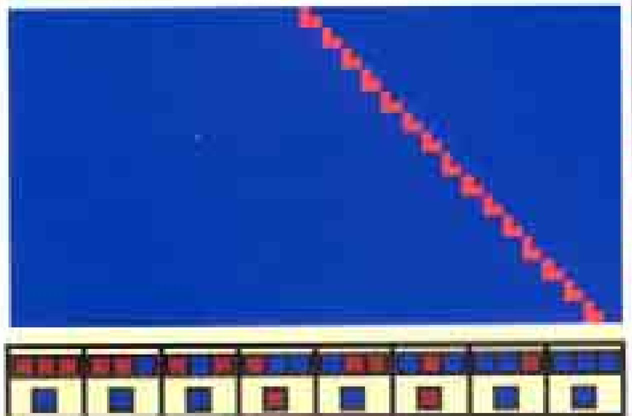
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [1 - |(-2u_{i-1} - 2u_i + u_{i+1} + 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 20

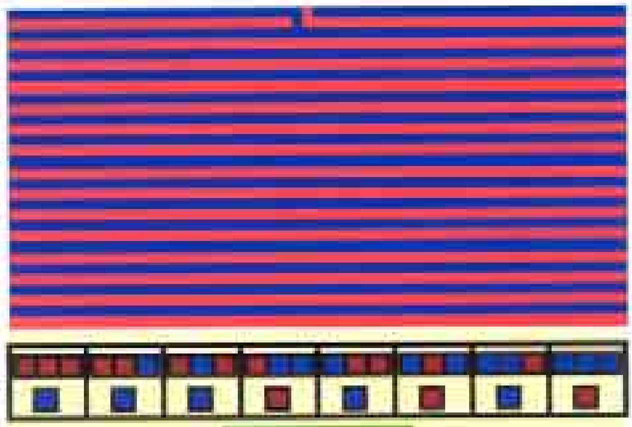
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} - u_i - 2u_{i+1} - 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 21

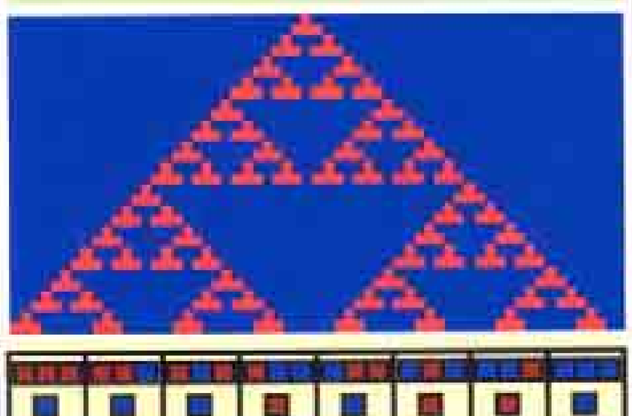
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [1 - |(u_{i-1} + u_i + u_{i+1} + 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 22

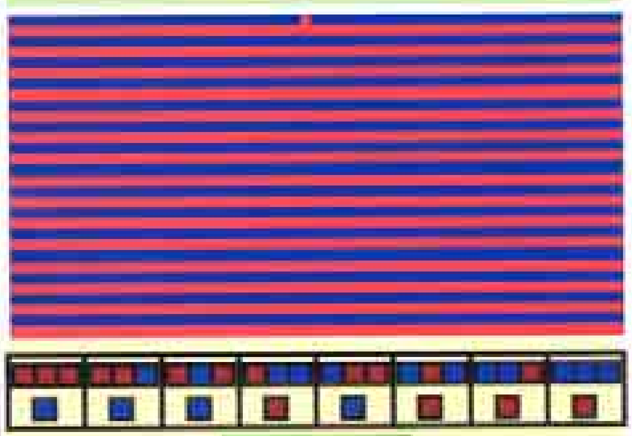
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} - u_i - u_{i+1})$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 23

Table 2. (*Continued*)

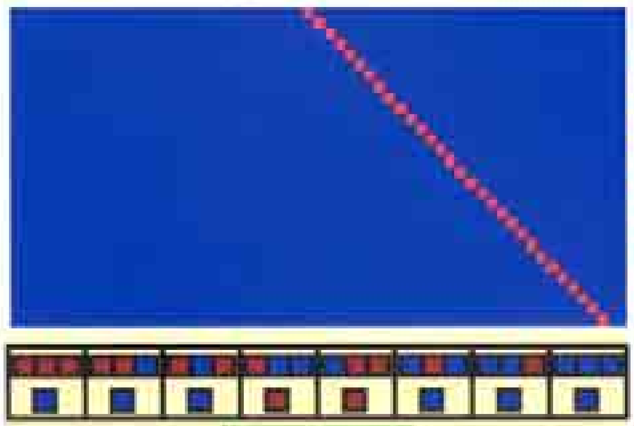
Rule 24 ($\kappa = 2$)

State Equation

$$\dot{x}_i = -x_i + 2y_i + [-2 + |(u_{i-1} - u_i - u_{i+1})|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 24

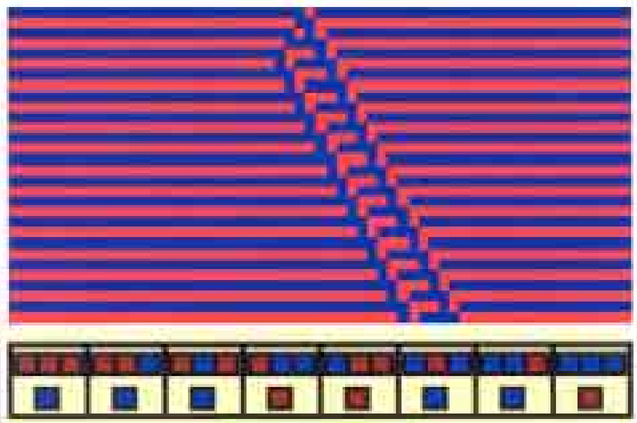
Rule 25 ($\kappa = 2$)

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(u_{i-1} - 2u_i + 4u_{i+1} + 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 25

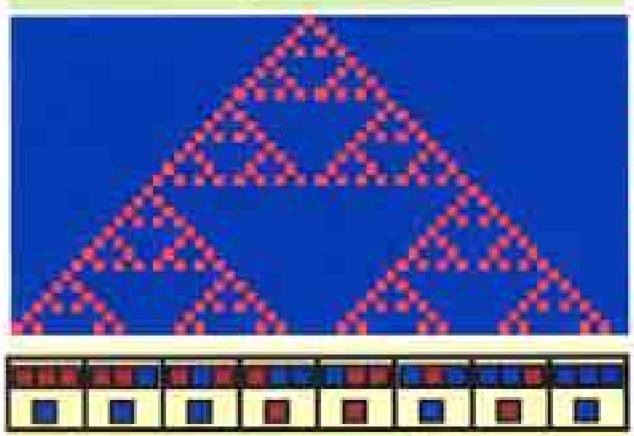
Rule 26 ($\kappa = 2$)

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(2u_{i-1} - u_i + 4u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 26

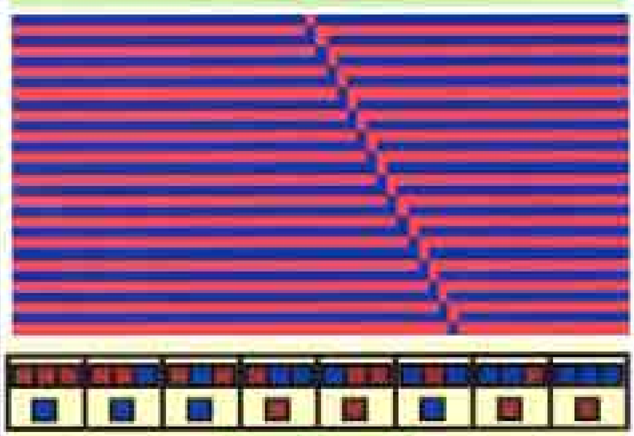
Rule 27 ($\kappa = 3$)

State Equation

$$\dot{x}_i = -x_i + 2y_i + \{2 - |[-3 + |(-2u_{i-1} - 4u_i + u_{i+1} - 3)|]|\}$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 27

Table 2. (*Continued*)

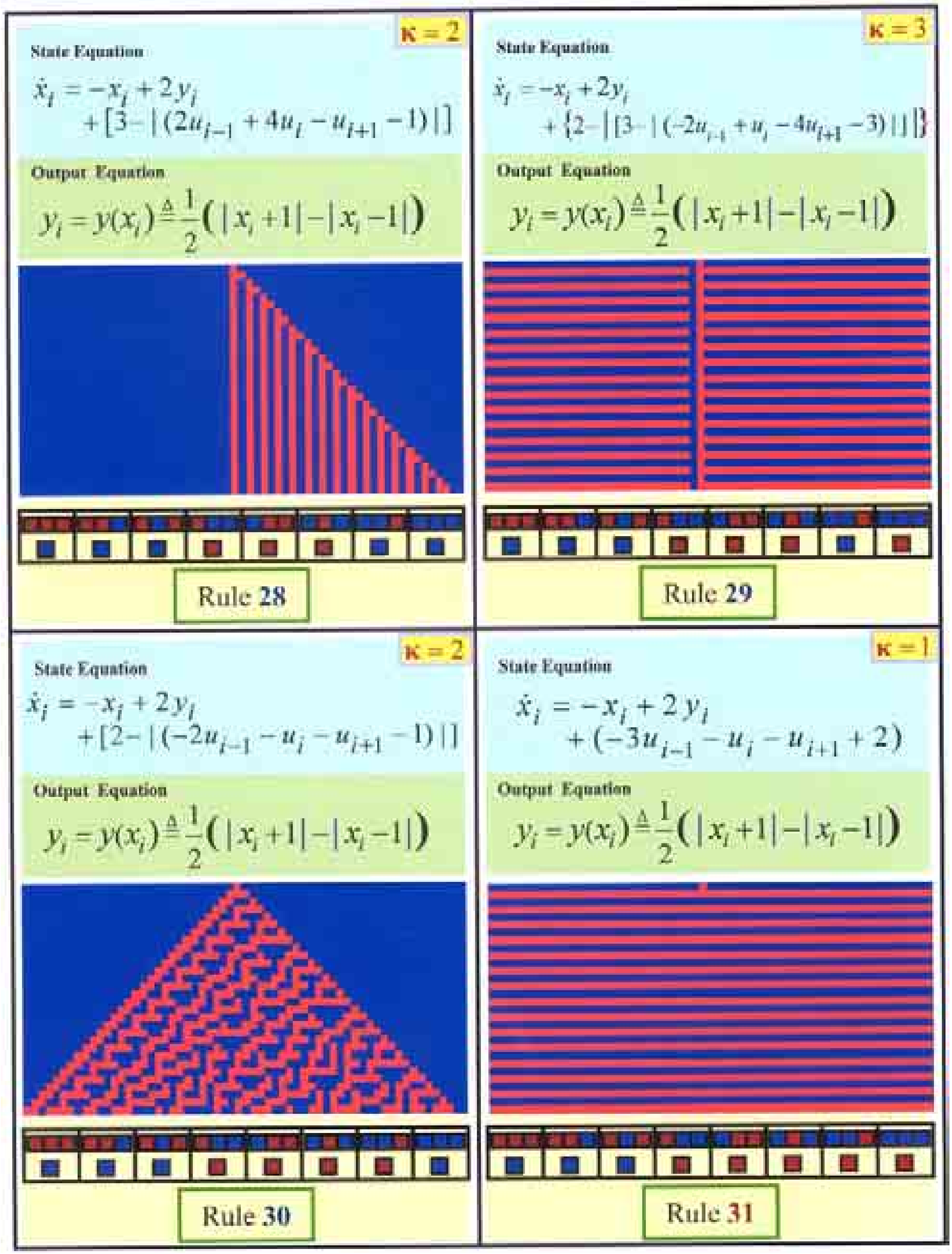


Table 2. (*Continued*)

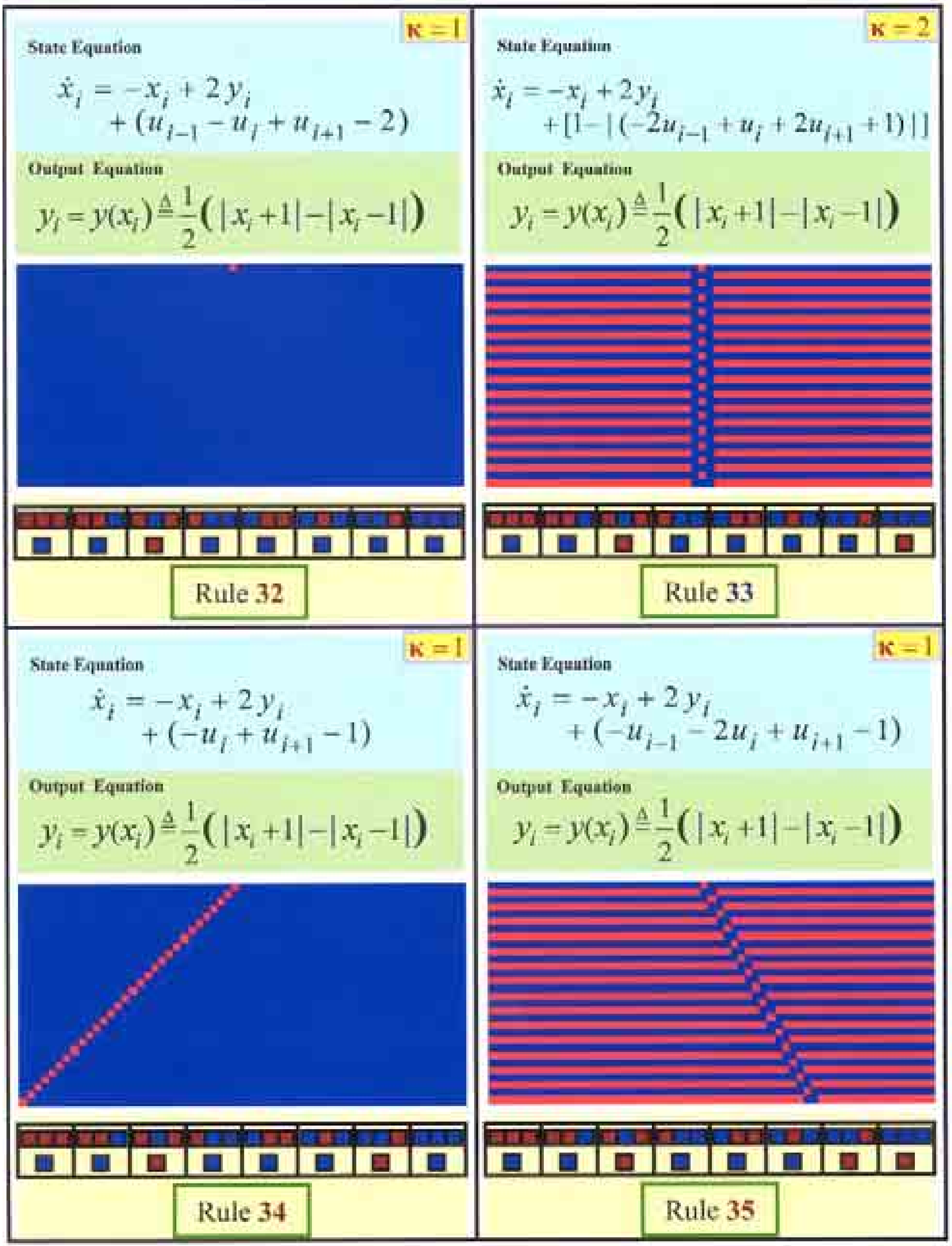


Table 2. (*Continued*)

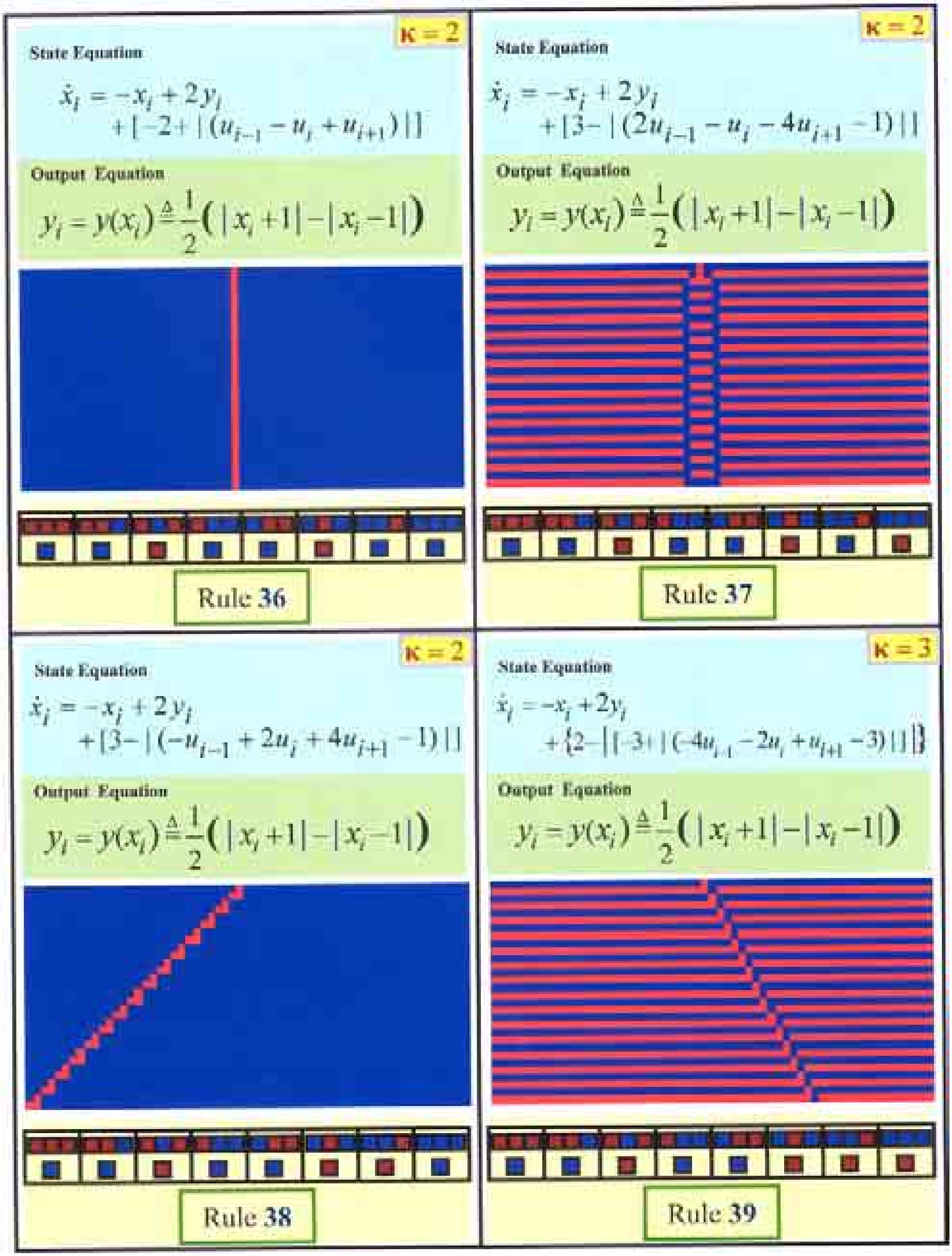


Table 2. (*Continued*)

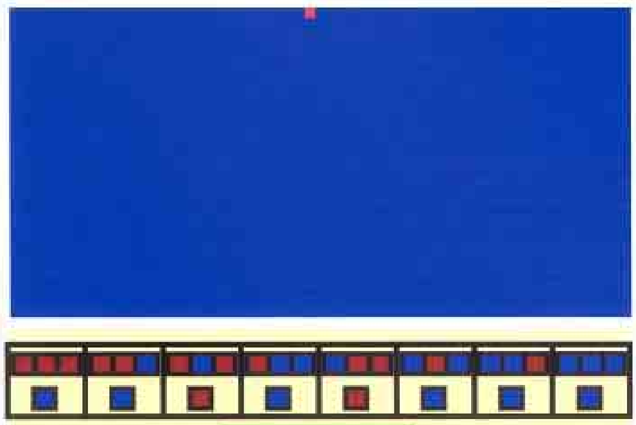
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [1 - |(-2u_{i-1} - 2u_i + u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **40**

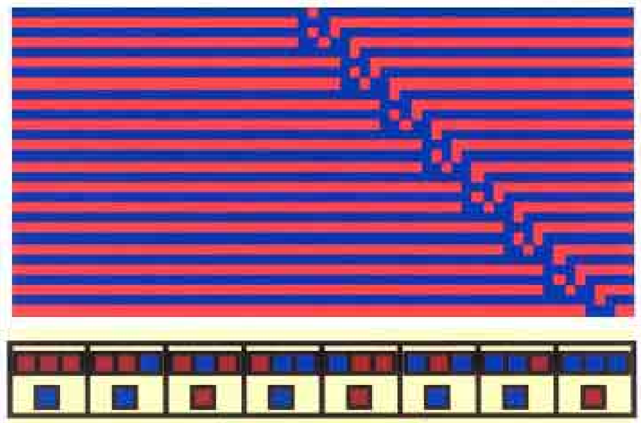
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [1 - |(-u_{i-1} - u_i + u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **41**

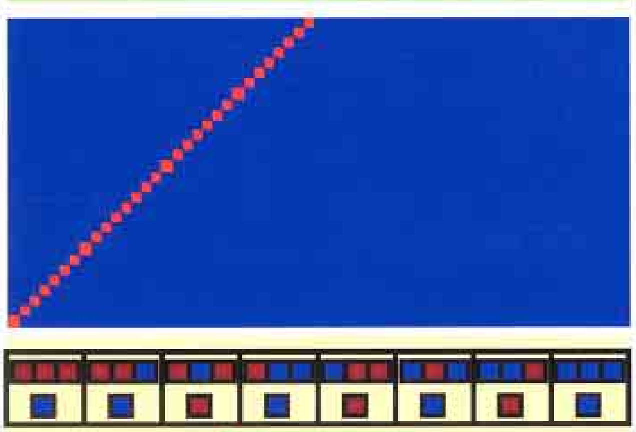
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} - u_i + 2u_{i+1} - 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **42**

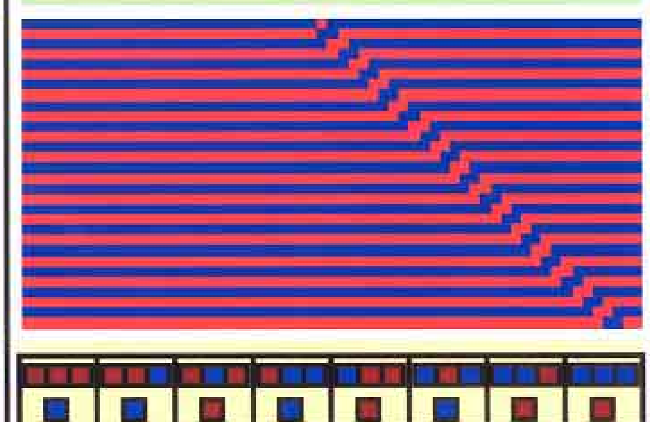
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} - u_i + u_{i+1})$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **43**

Table 2. (*Continued*)

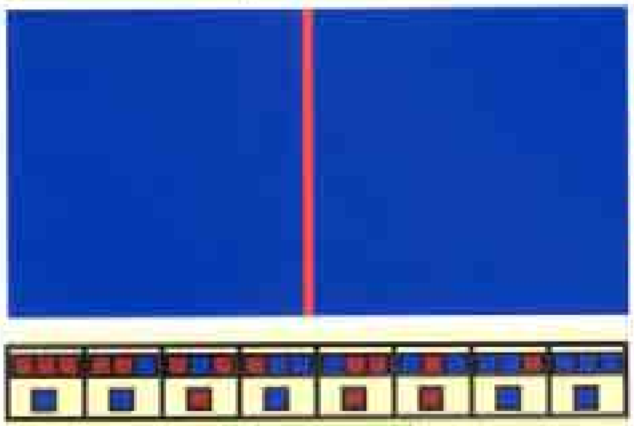
κ = 2

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(-4u_{i-1} - 2u_i + u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **44**

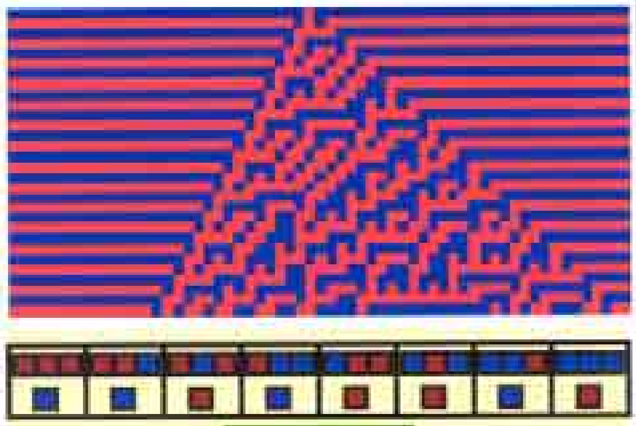
κ = 2

State Equation

$$\dot{x}_i = -x_i + 2y_i + [2 - |(-2u_{i-1} - u_i + u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **45**

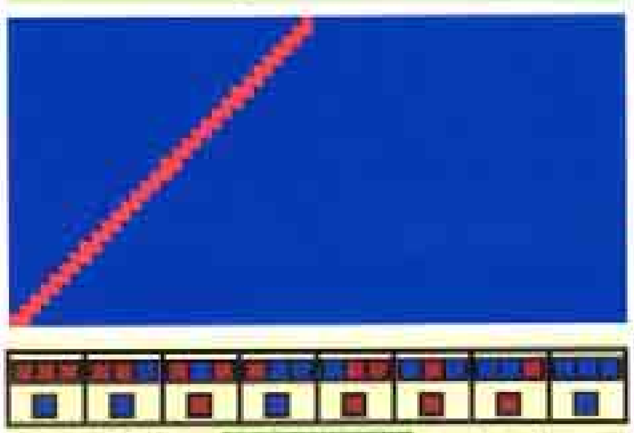
κ = 3

State Equation

$$\dot{x}_i = -x_i + 2y_i + \{2 - |[3 - |(u_{i-1} - 2u_i - 3u_{i+1} + 4)|]|\}$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **46**

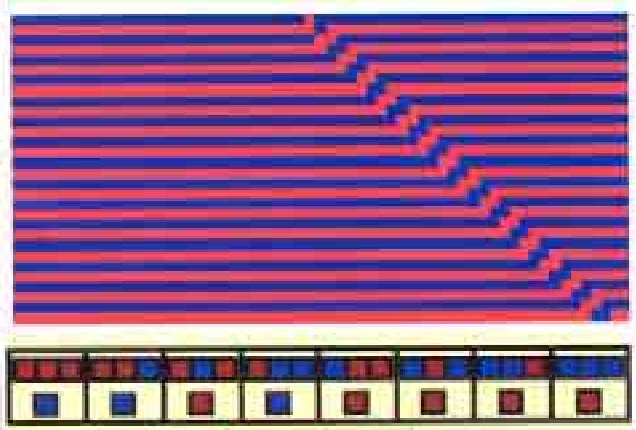
κ = 1

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-3u_{i-1} - u_i + u_{i+1} + 2)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **47**

Table 2. (*Continued*)

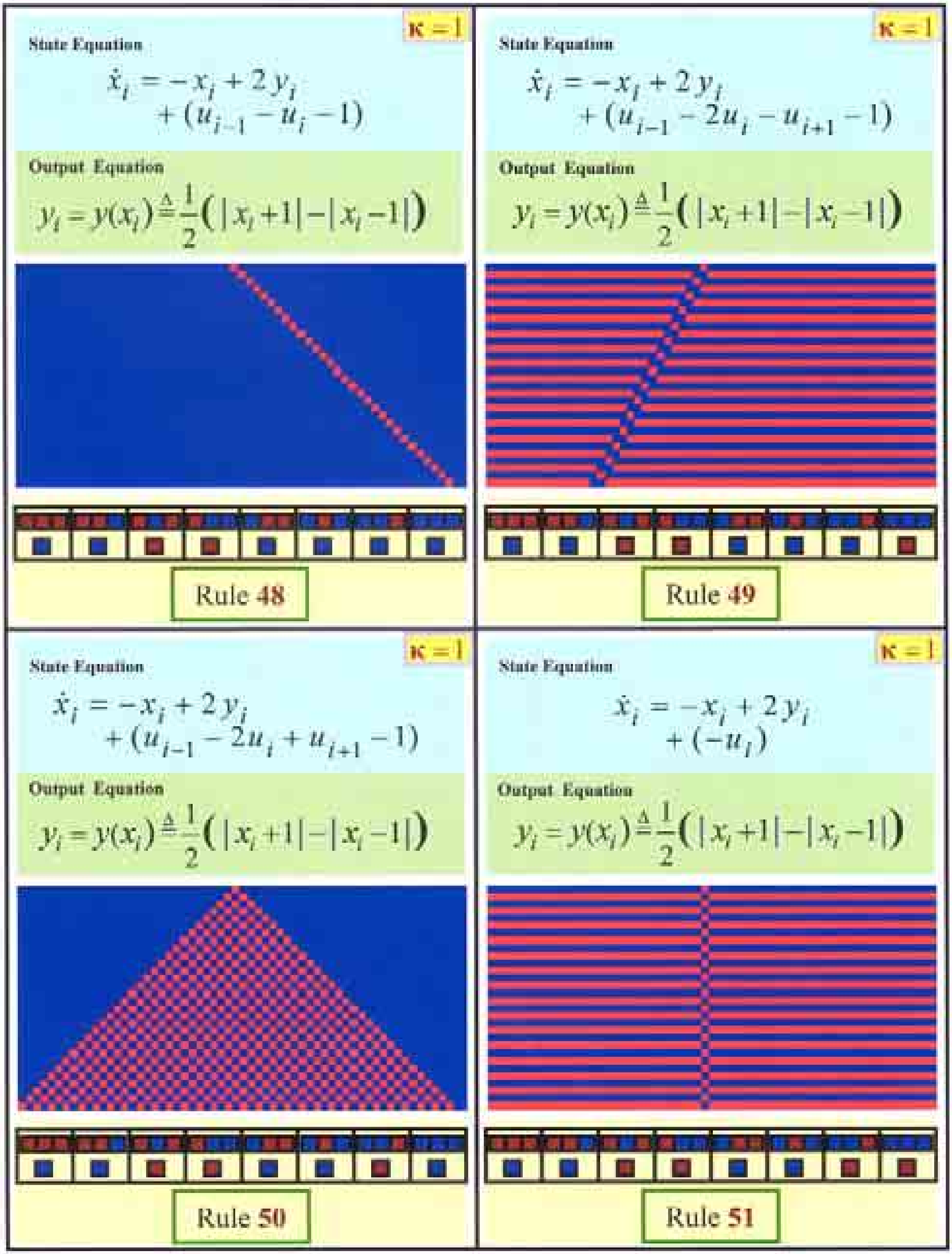


Table 2. (*Continued*)

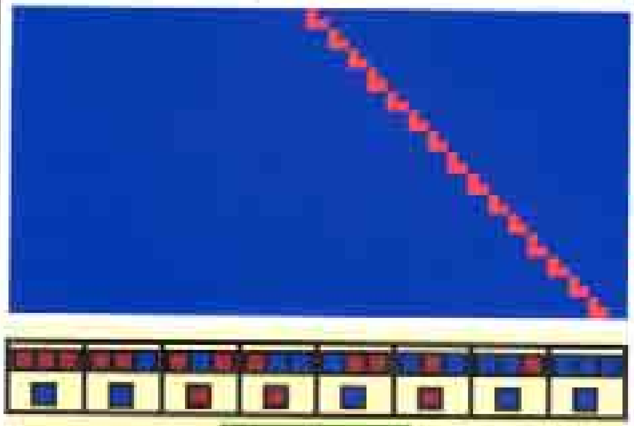
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(-2u_{i-1} - 4u_i - u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **52**

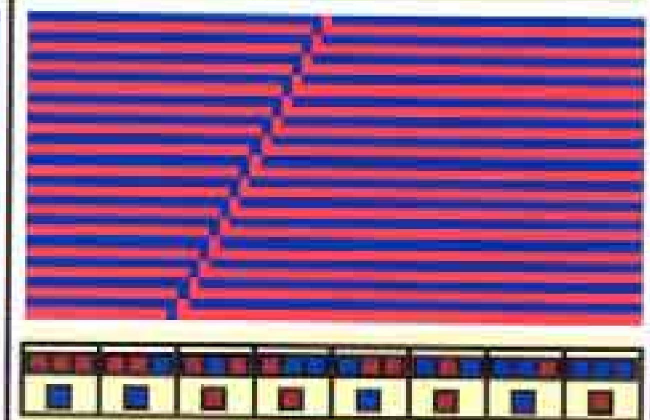
$\kappa = 3$

State Equation

$$\dot{x}_i = -x_i + 2y_i + \{2 - |[-3 + |(4u_{i-1} + u_i - 3u_{i+1} - 2)|]|\}$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **53**

$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [2 - |(-u_{i-1} - 2u_i - u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$

Rule **54**

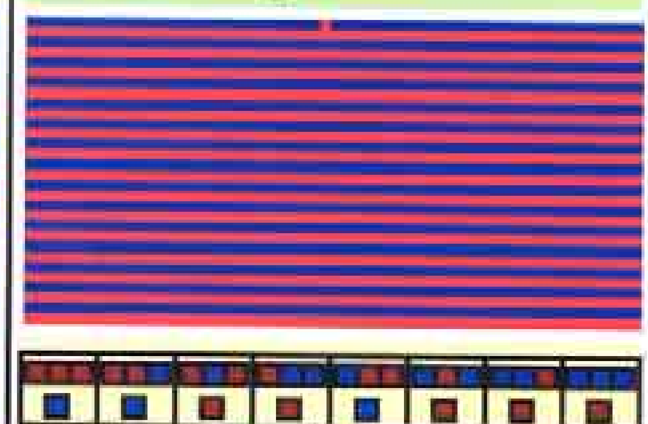
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} - 3u_i - u_{i+1} + 2)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **55**

Table 2. (*Continued*)

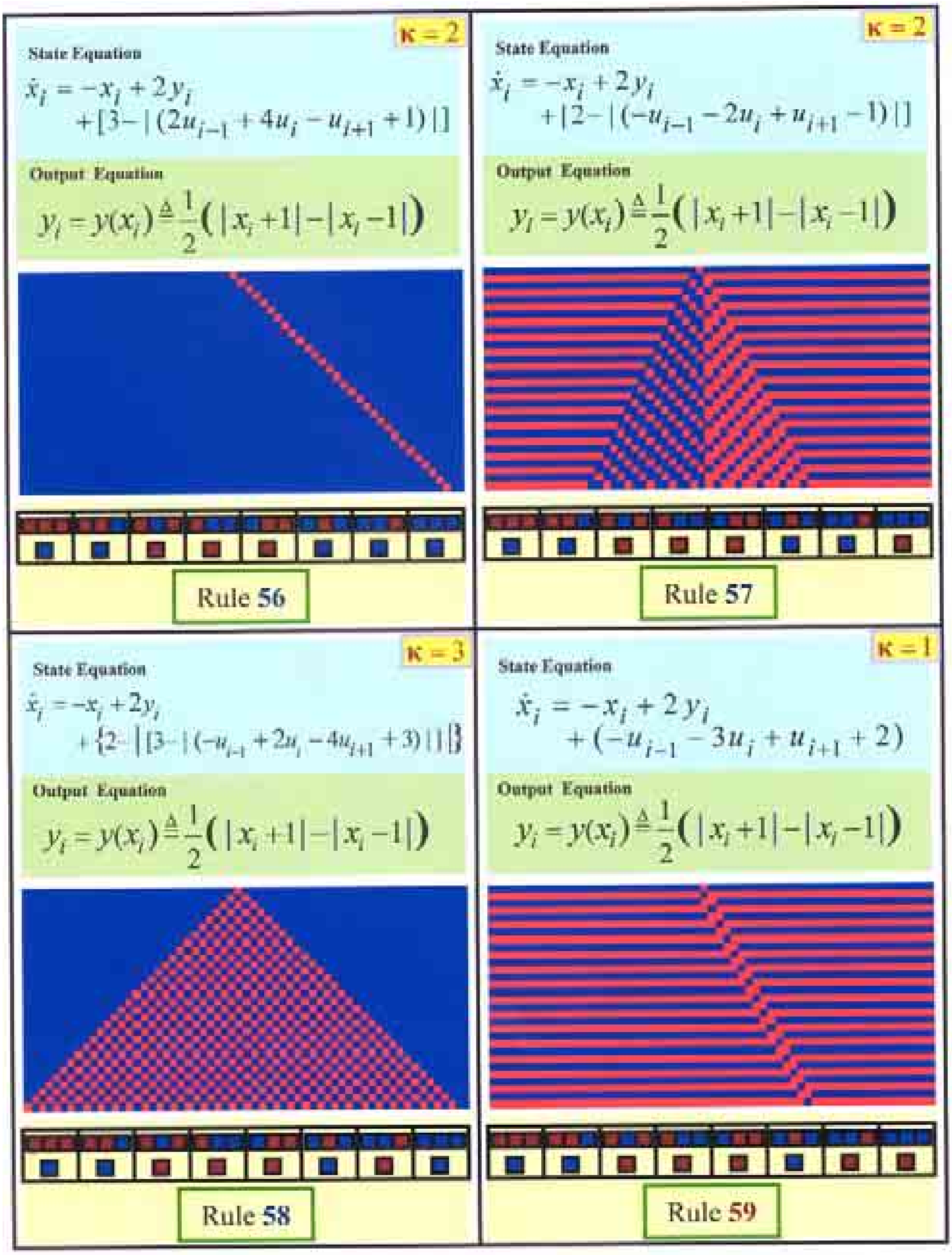


Table 2. (*Continued*)

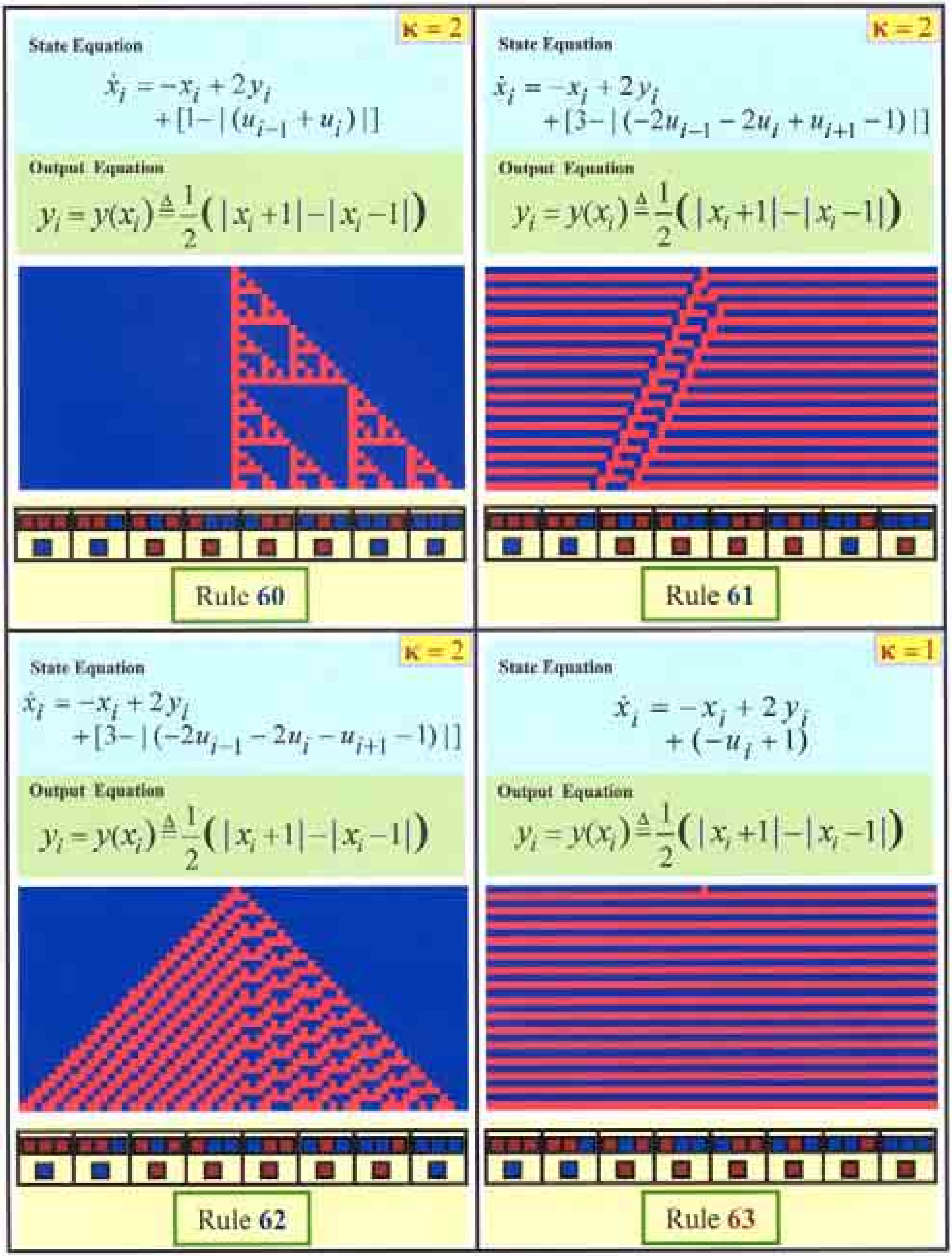


Table 2. (*Continued*)

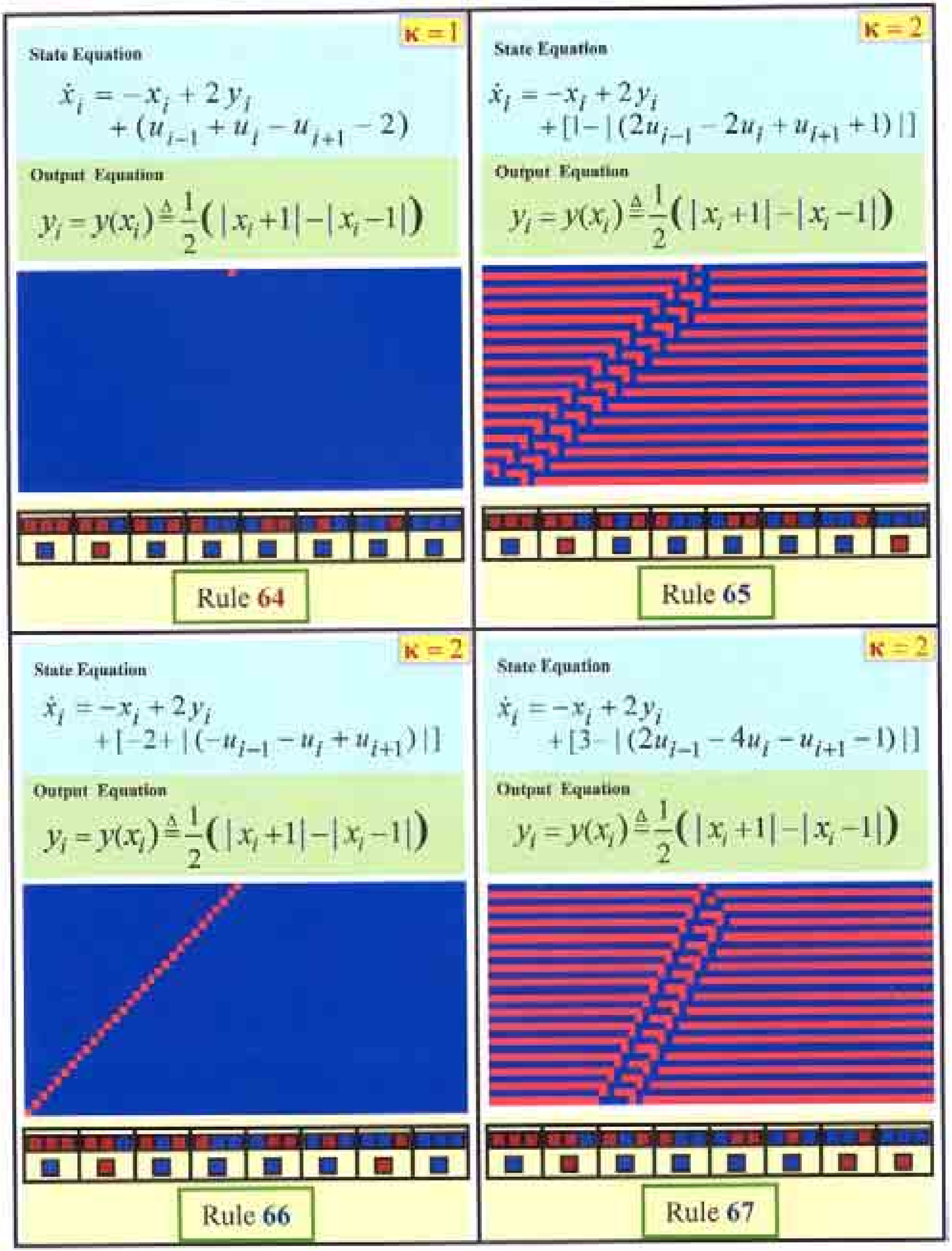


Table 2. (*Continued*)

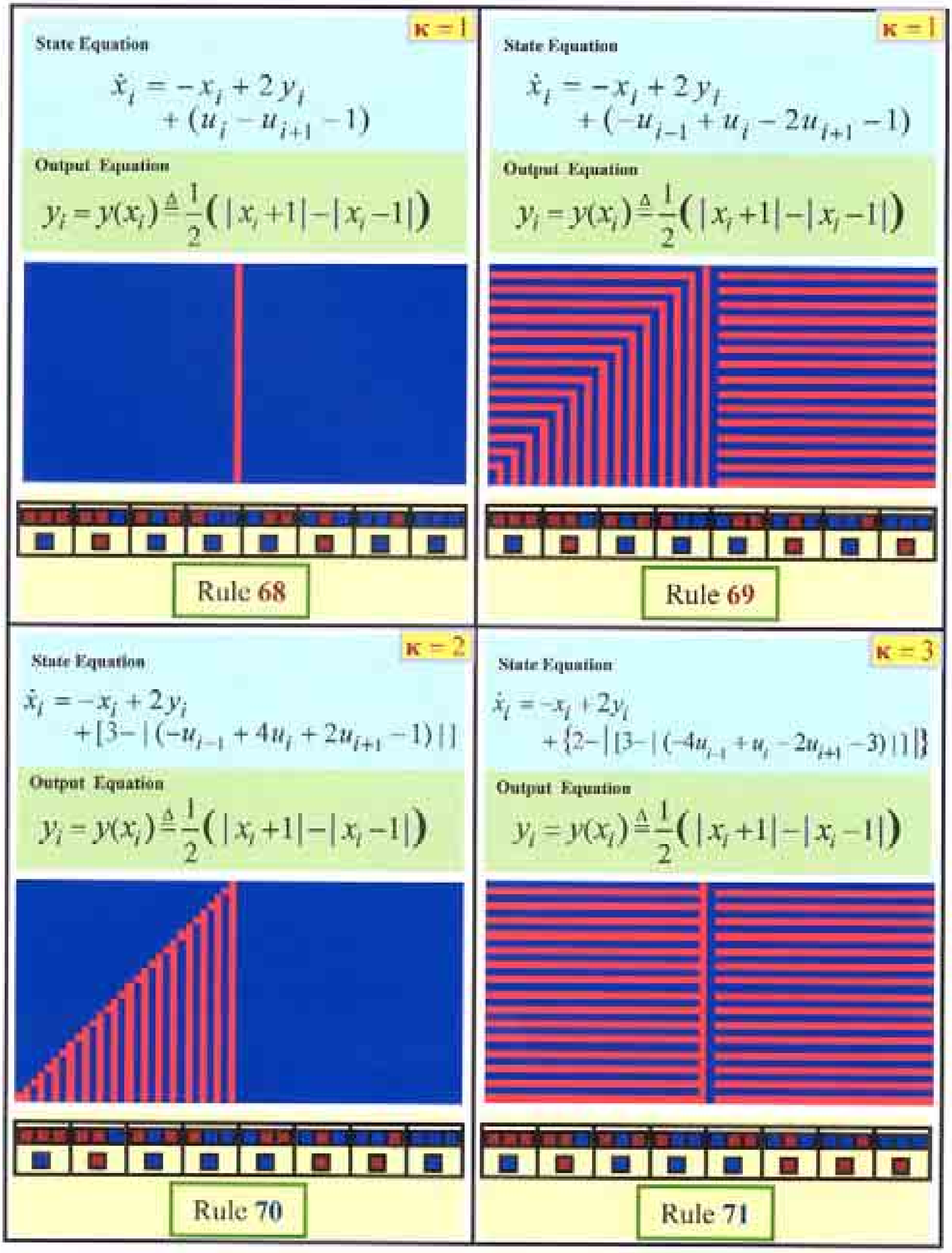


Table 2. (*Continued*)

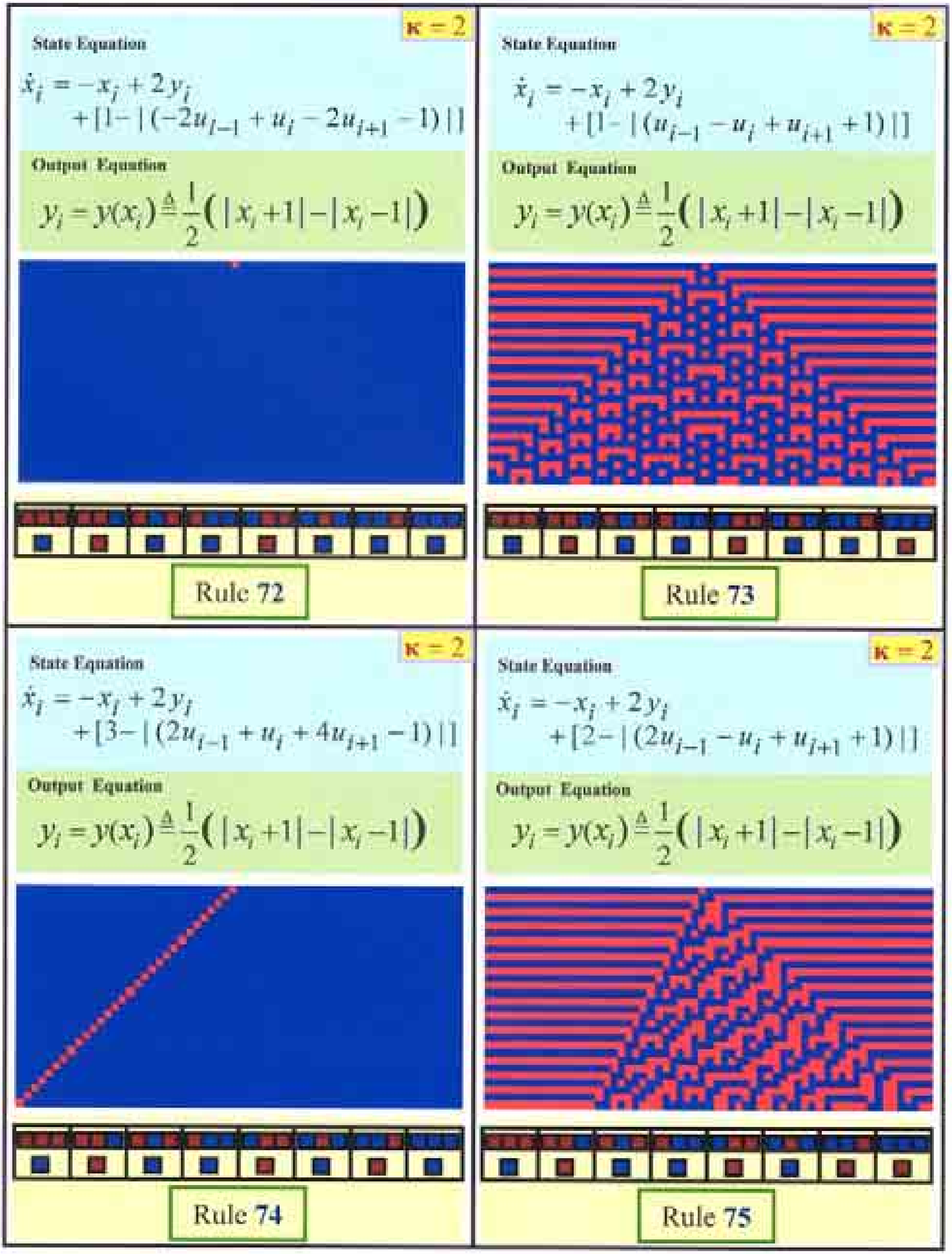


Table 2. (*Continued*)

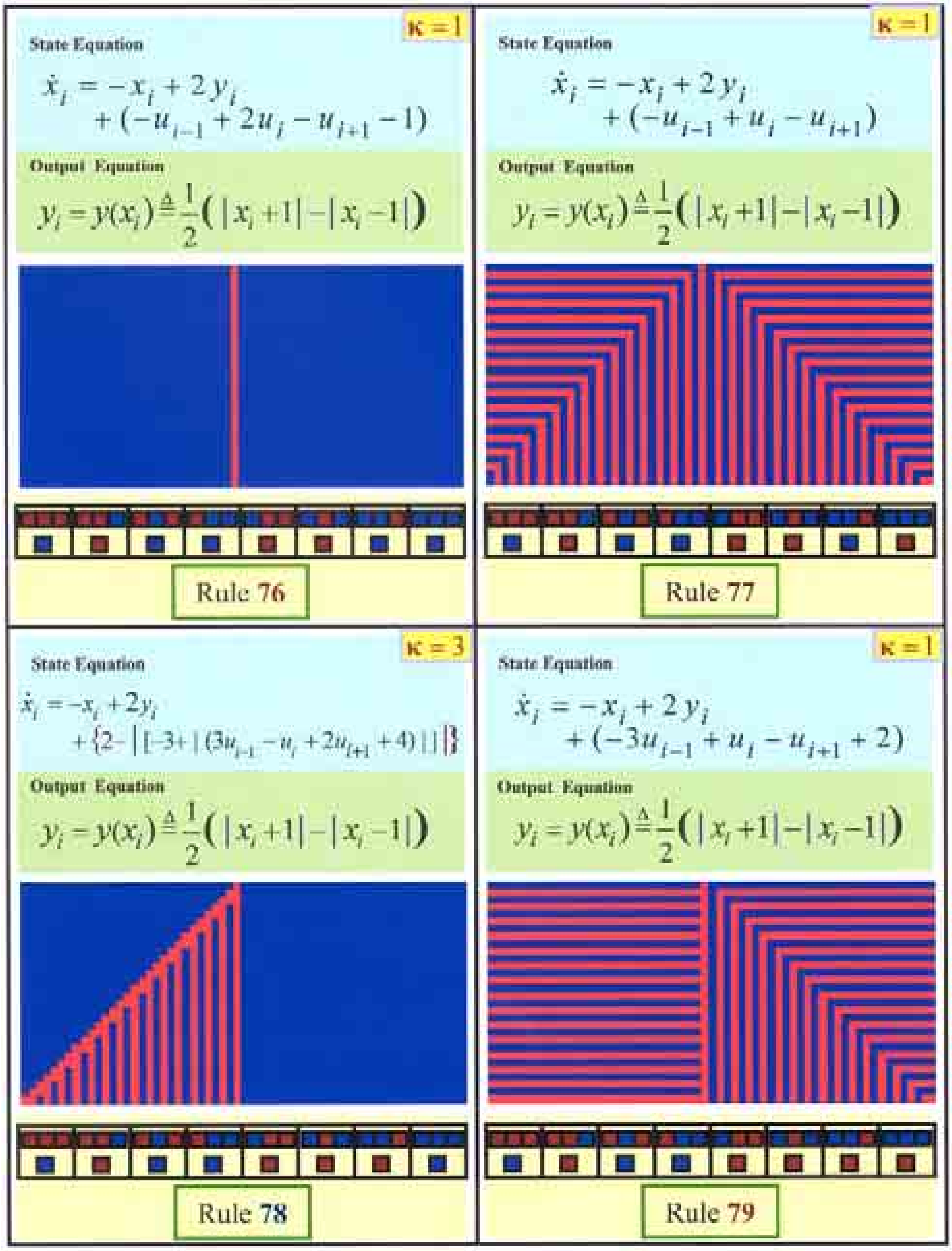


Table 2. (*Continued*)

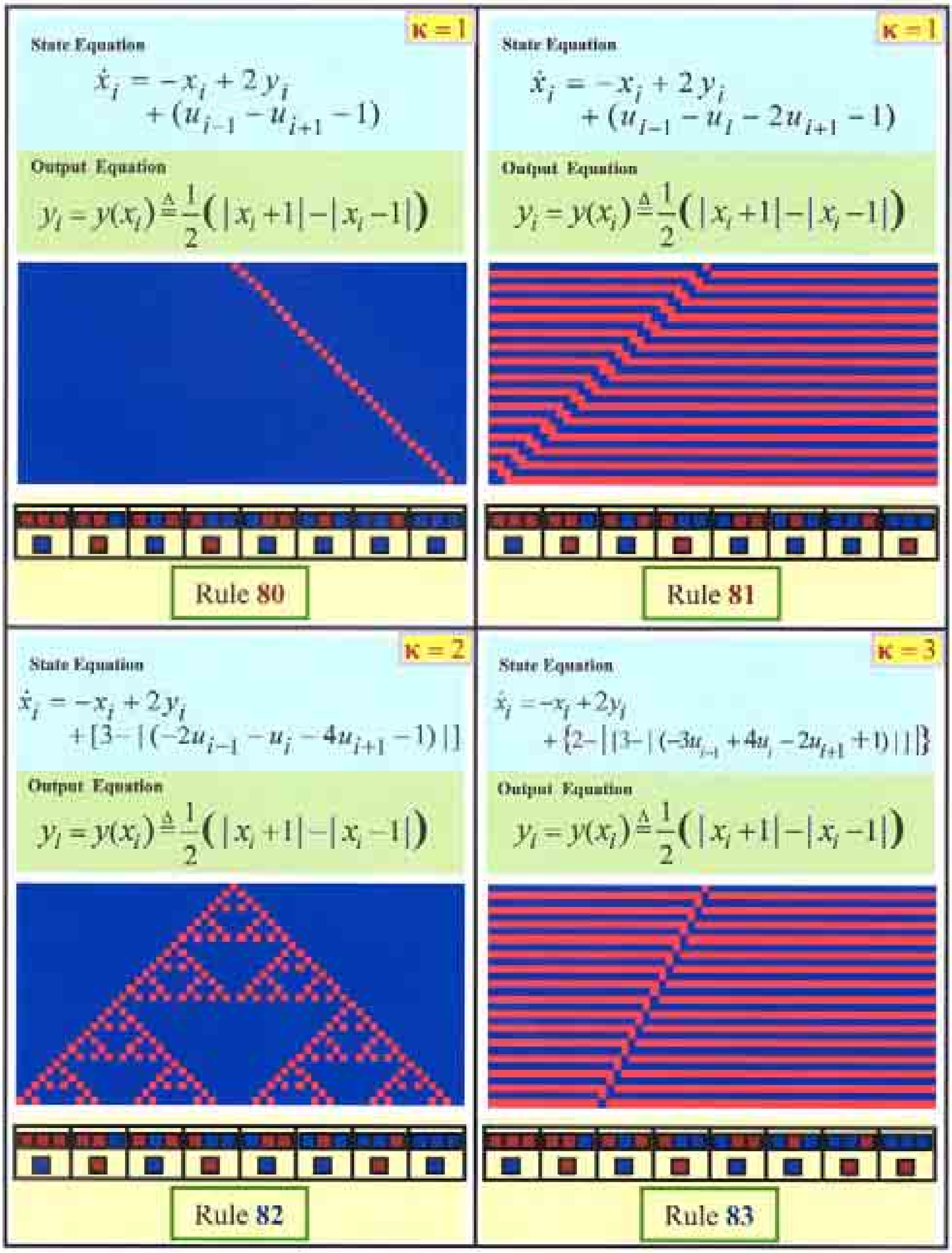


Table 2. (*Continued*)

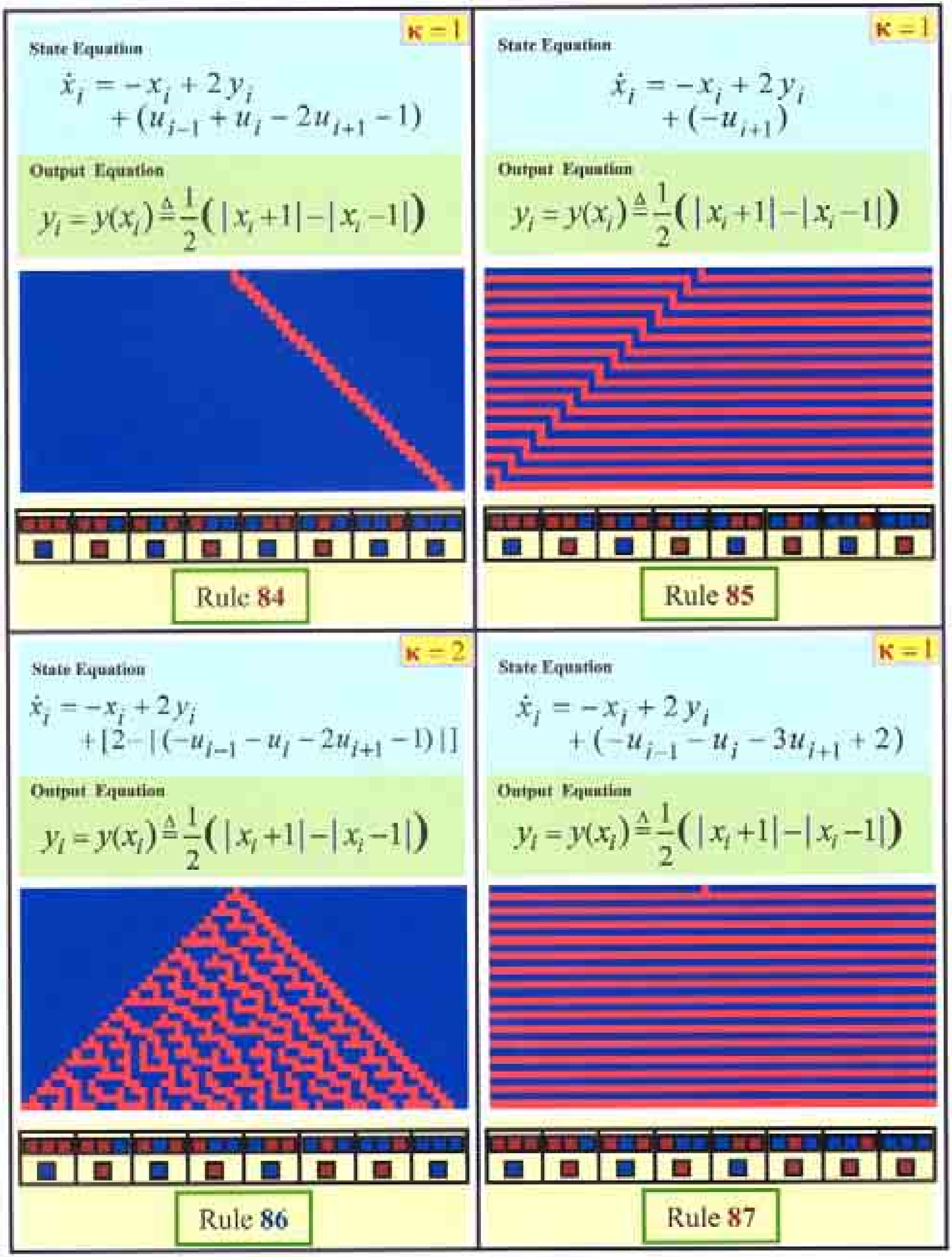


Table 2. (*Continued*)

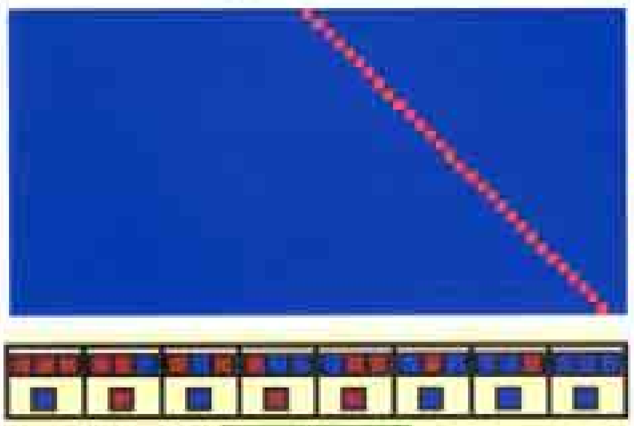
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(-4u_{i-1} - u_i - 2u_{i+1} + 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **88**

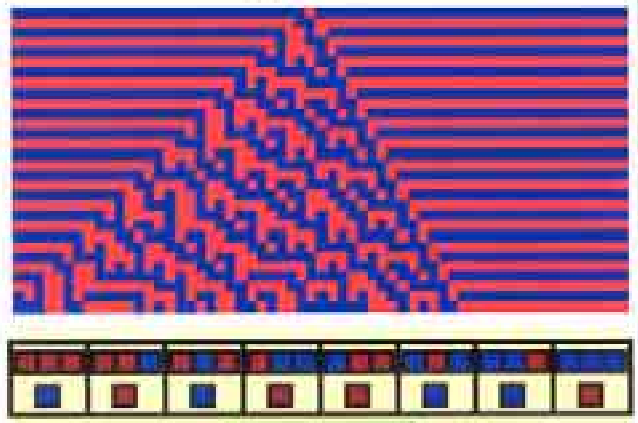
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [2 - |(u_{i-1} - u_i + 2u_{i+1} + 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **89**

$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [1 - |(u_{i-1} + u_{i+1})|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$

Rule **90**

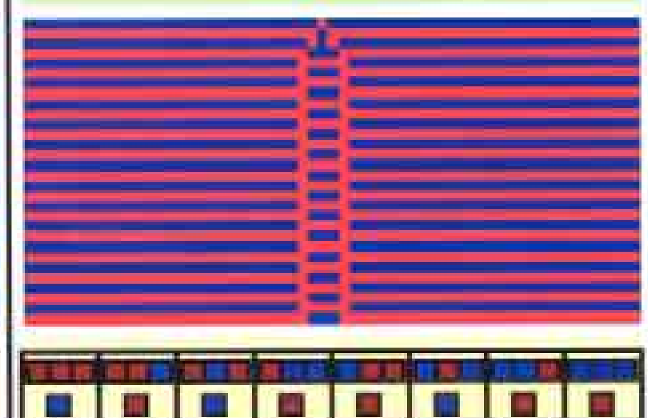
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(-2u_{i-1} + u_i - 2u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **91**

Table 2. (*Continued*)

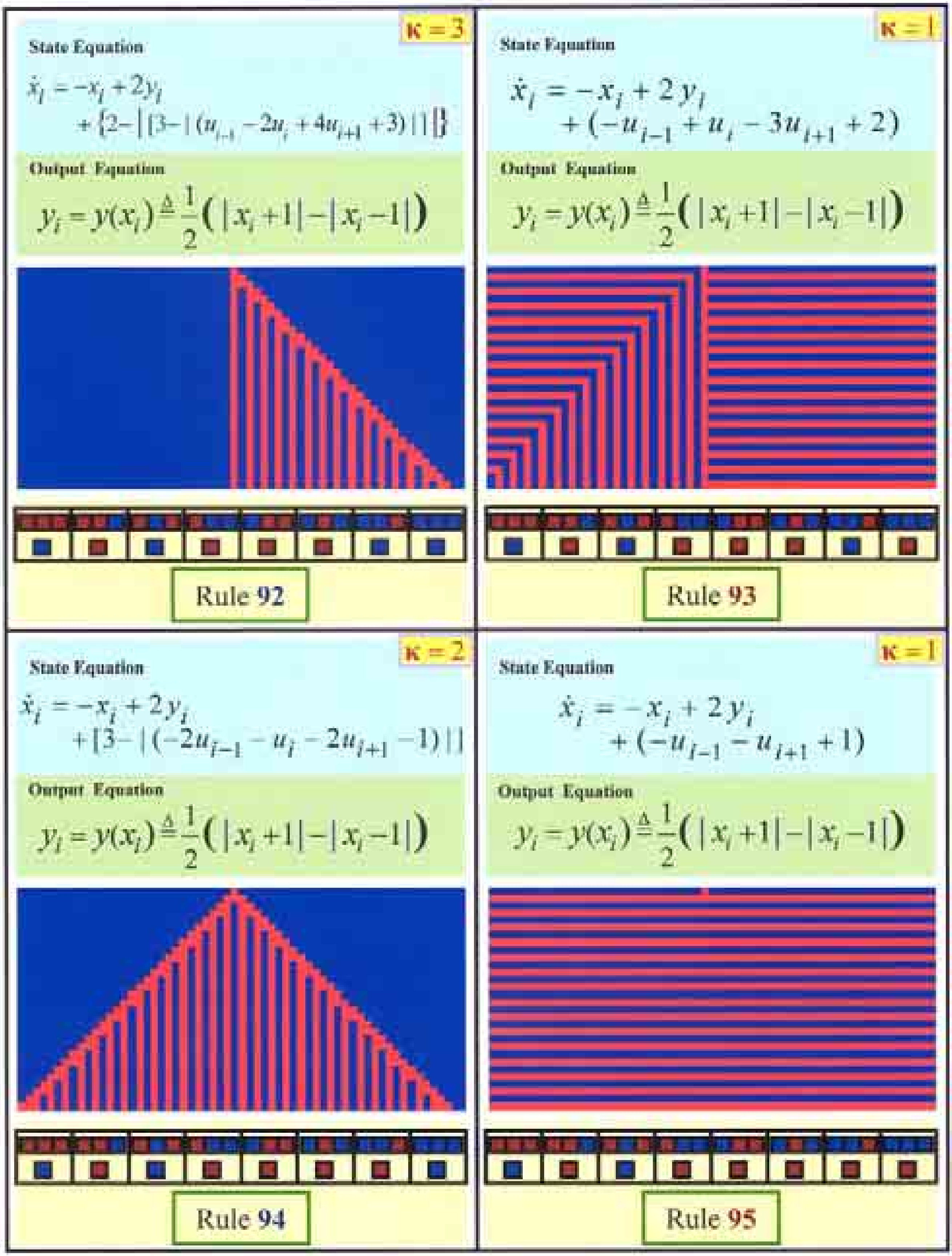


Table 2. (*Continued*)

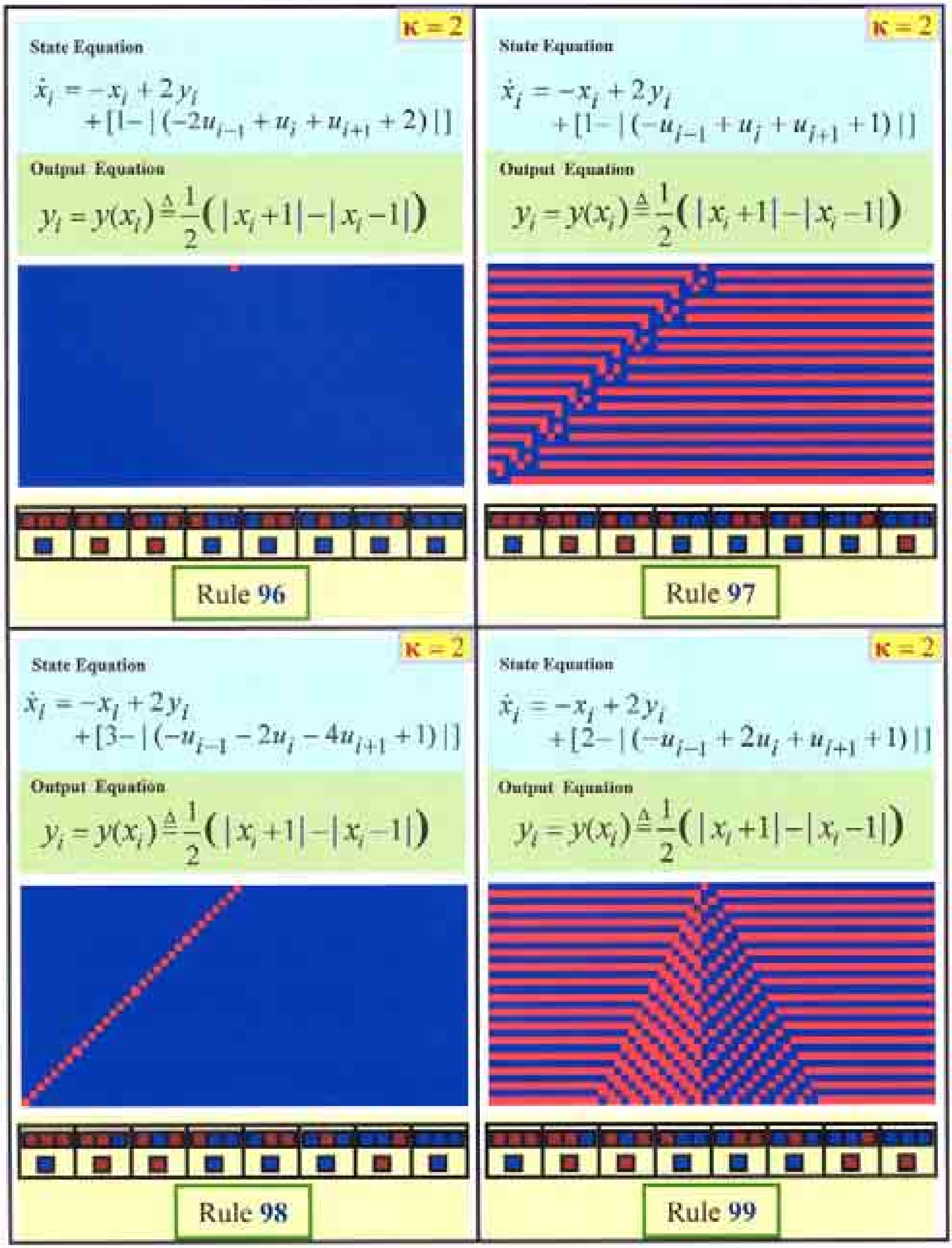


Table 2. (*Continued*)

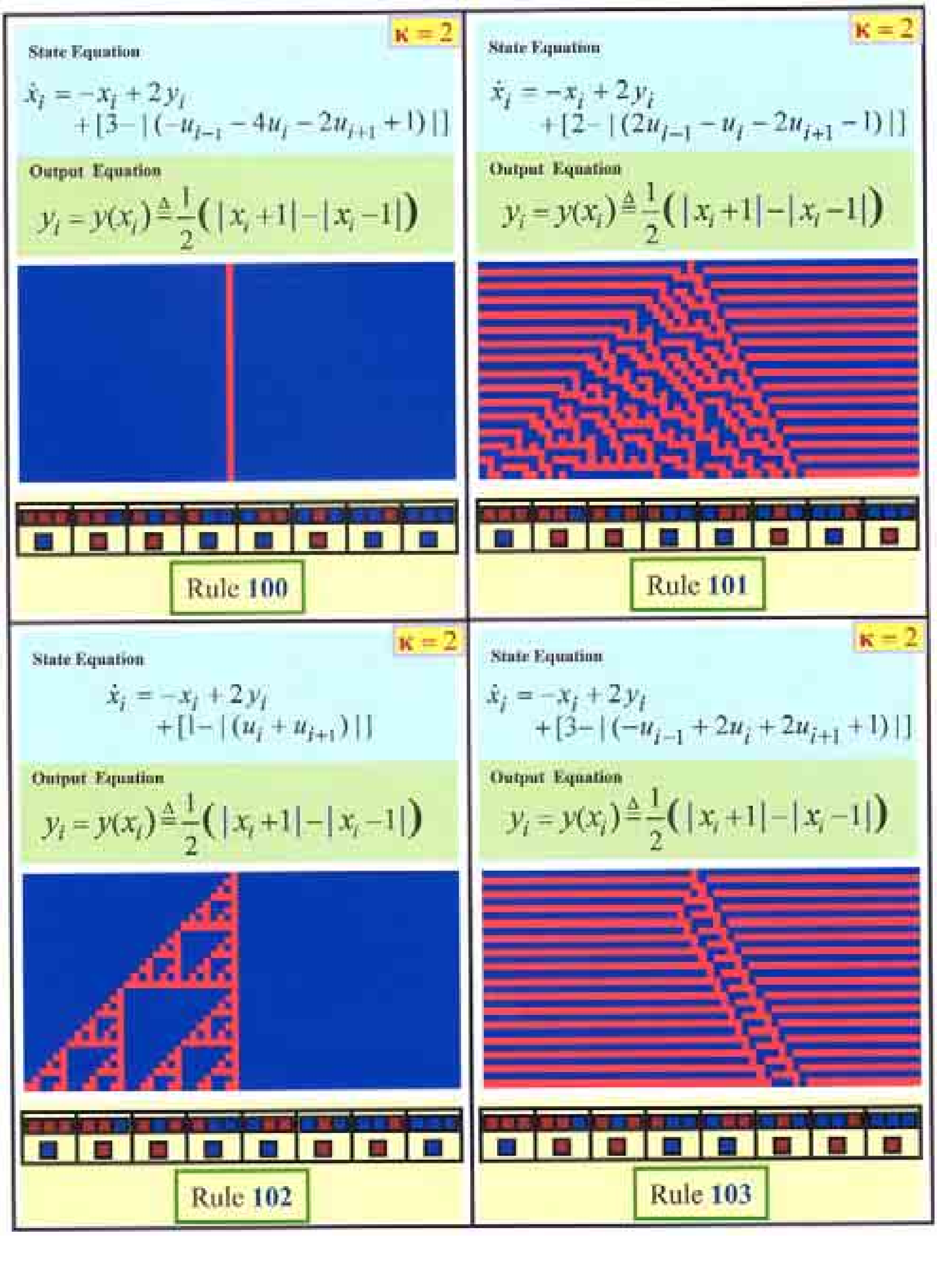


Table 2. (*Continued*)

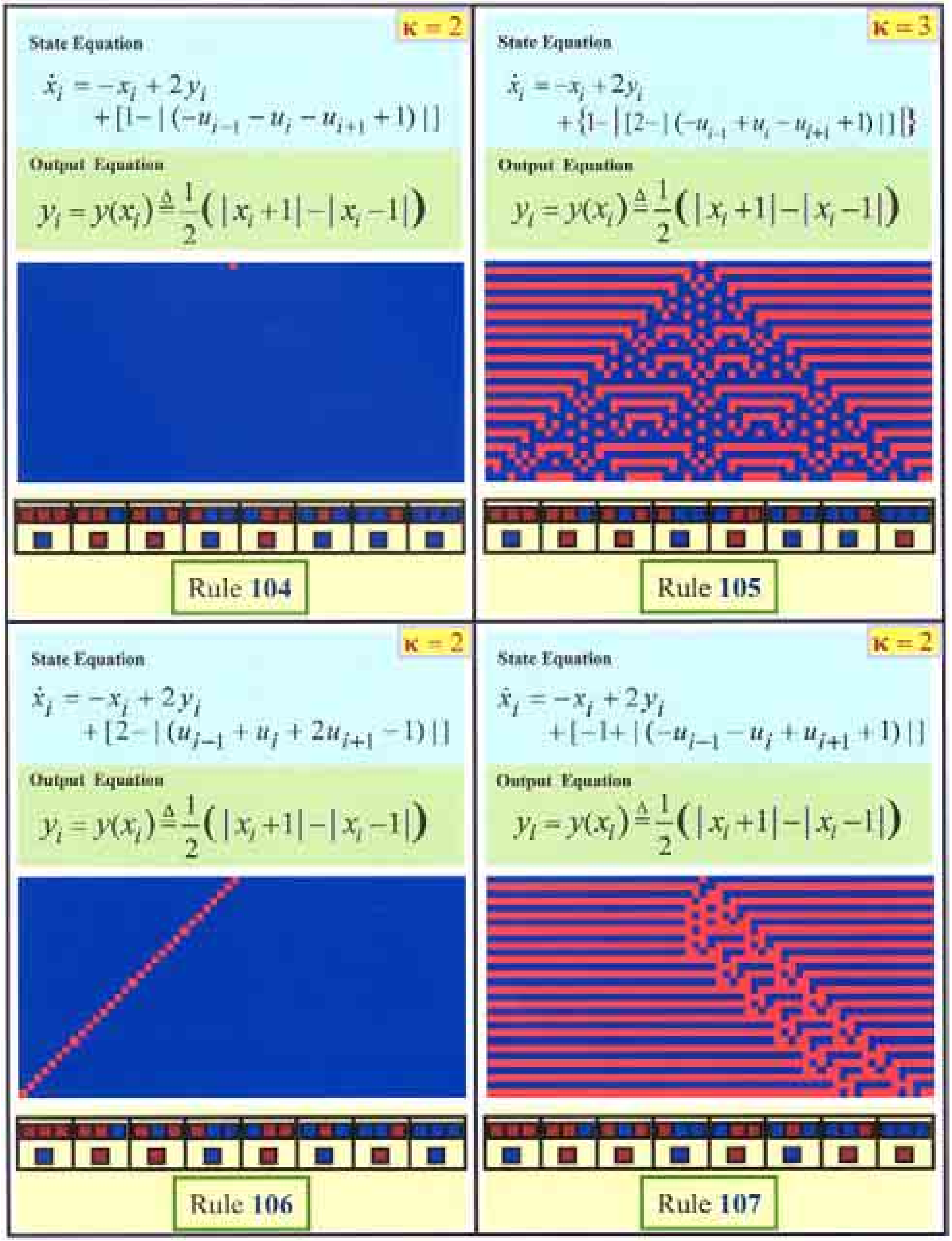


Table 2. (*Continued*)

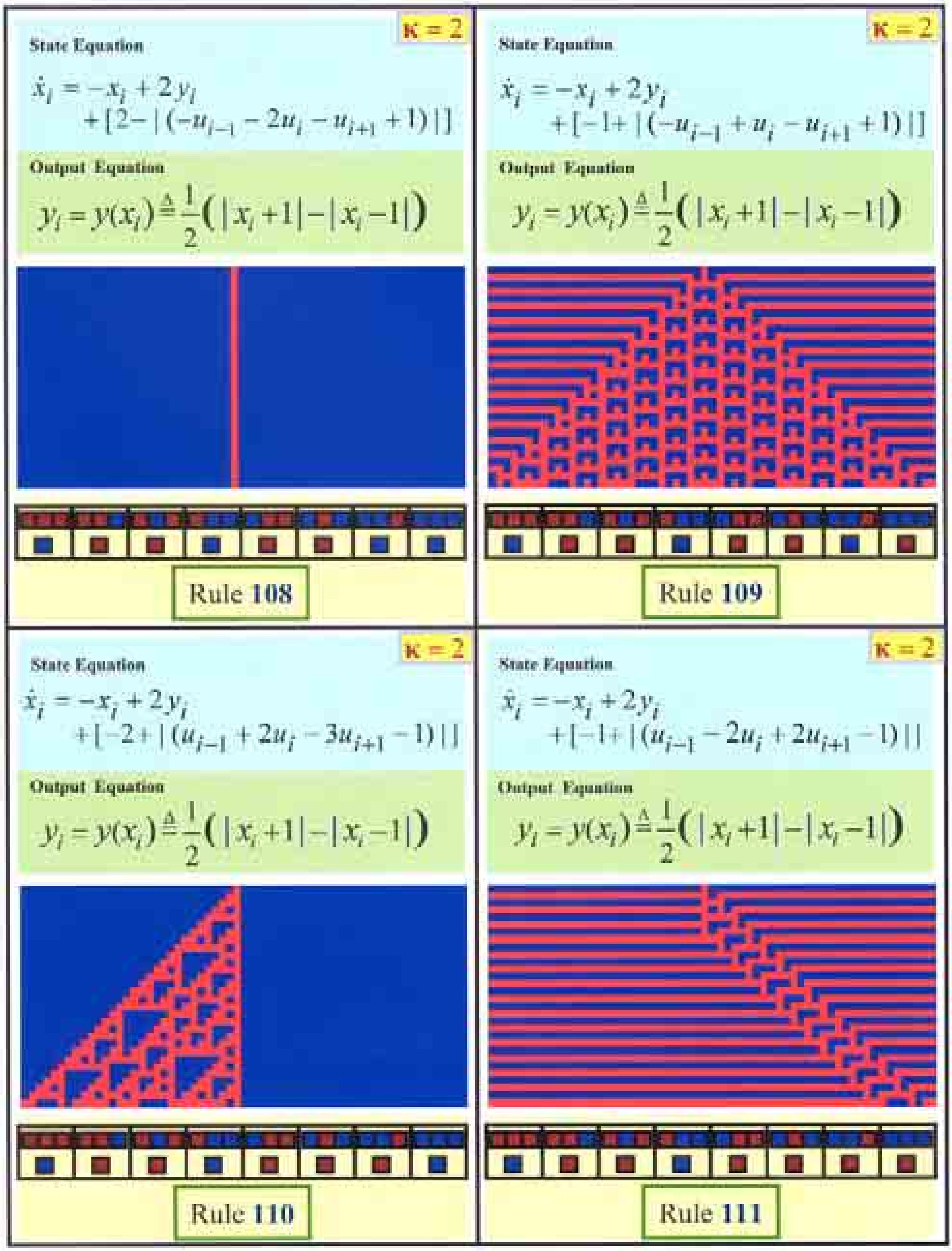


Table 2. (*Continued*)

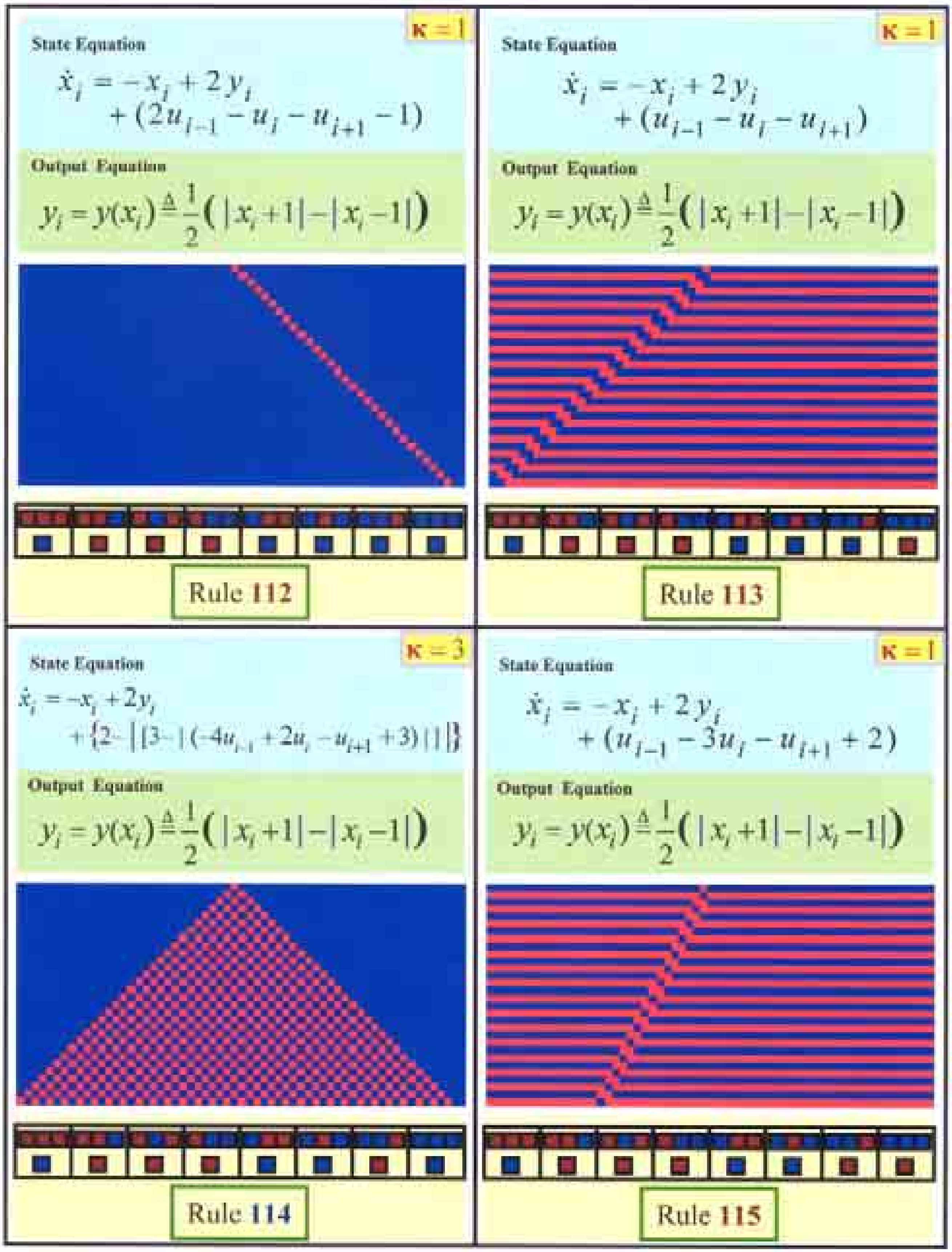


Table 2. (*Continued*)

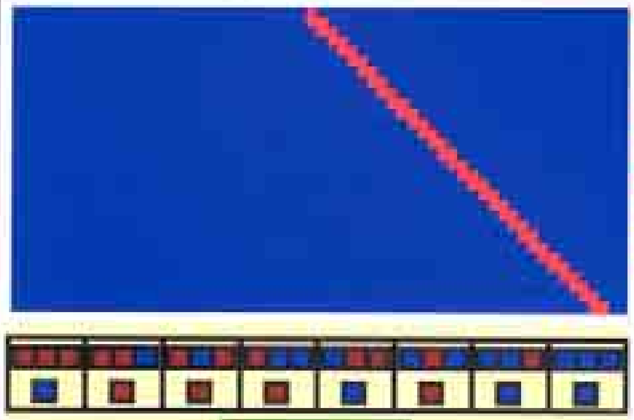
$\kappa = 3$

State Equation

$$\dot{x}_i = -x_i + 2y_i + \{2 - |[3 - |(-4u_{i-1} - u_i + 2u_{i+1} + 3)|]|\}$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **116**

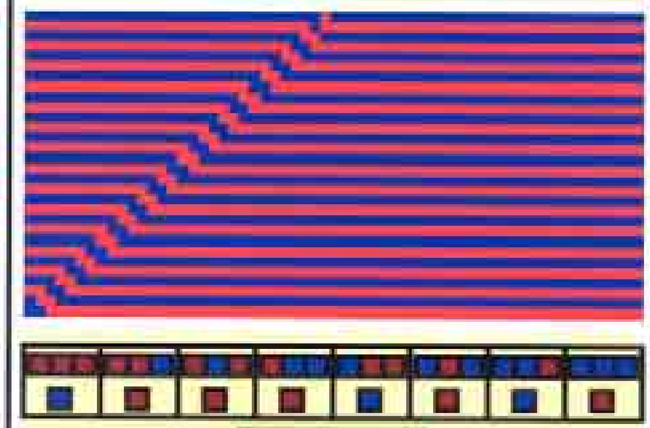
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (u_{i-1} - u_i - 3u_{i+1} + 2)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **117**

$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(-u_{i-1} - 2u_i - 2u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$

Rule **118**

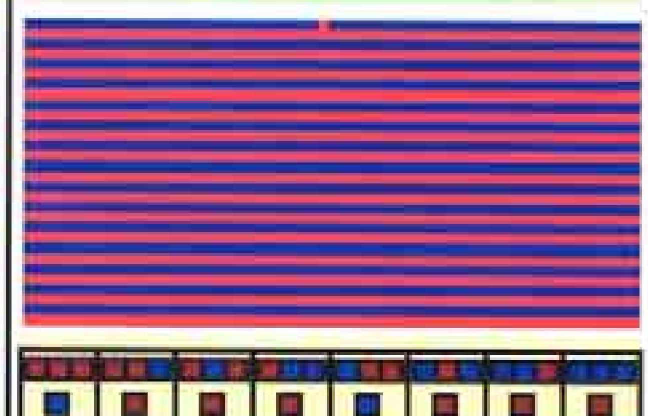
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_i - u_{i+1} + 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **119**

Table 2. (*Continued*)

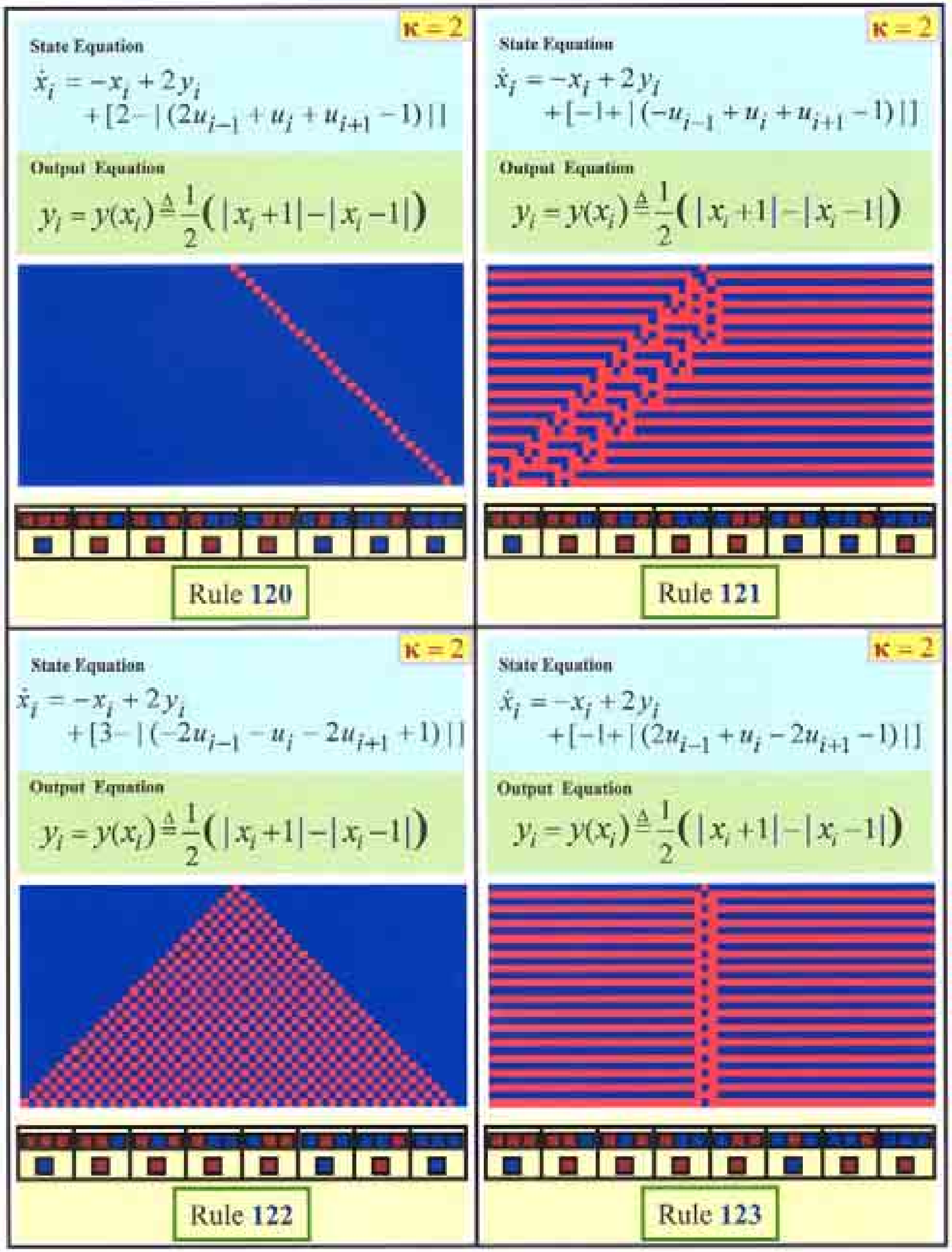


Table 2. (*Continued*)

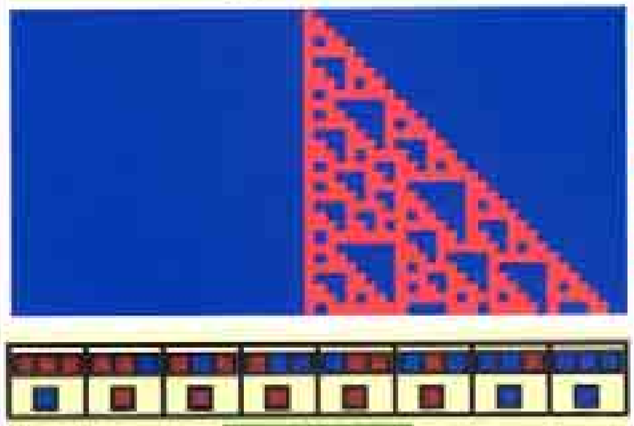
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(2u_{i-1} + 2u_i + u_{i+1} - 1)||$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}\left(|x_i + 1| - |x_i - 1|\right)$$



Rule **124**

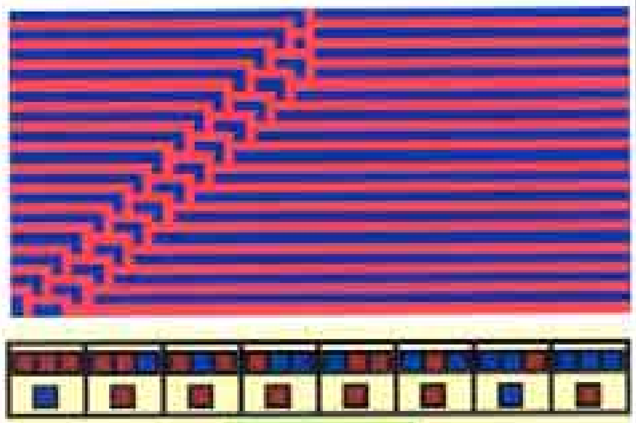
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [-1 + |(-u_{i-1} + u_i + 2u_{i+1} - 2)||$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}\left(|x_i + 1| - |x_i - 1|\right)$$



Rule **125**

$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [2 - |(-u_{i-1} - u_i - u_{i+1})|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}\left(|x_i + 1| - |x_i - 1|\right)$$

Rule **126**

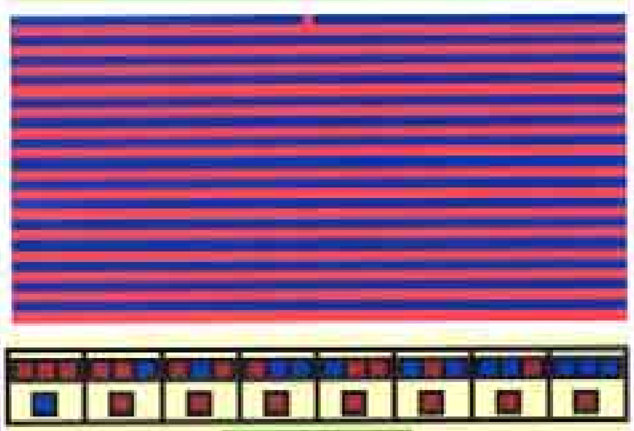
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} - u_i - u_{i+1} + 2)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}\left(|x_i + 1| - |x_i - 1|\right)$$



Rule **127**

Table 2. (*Continued*)

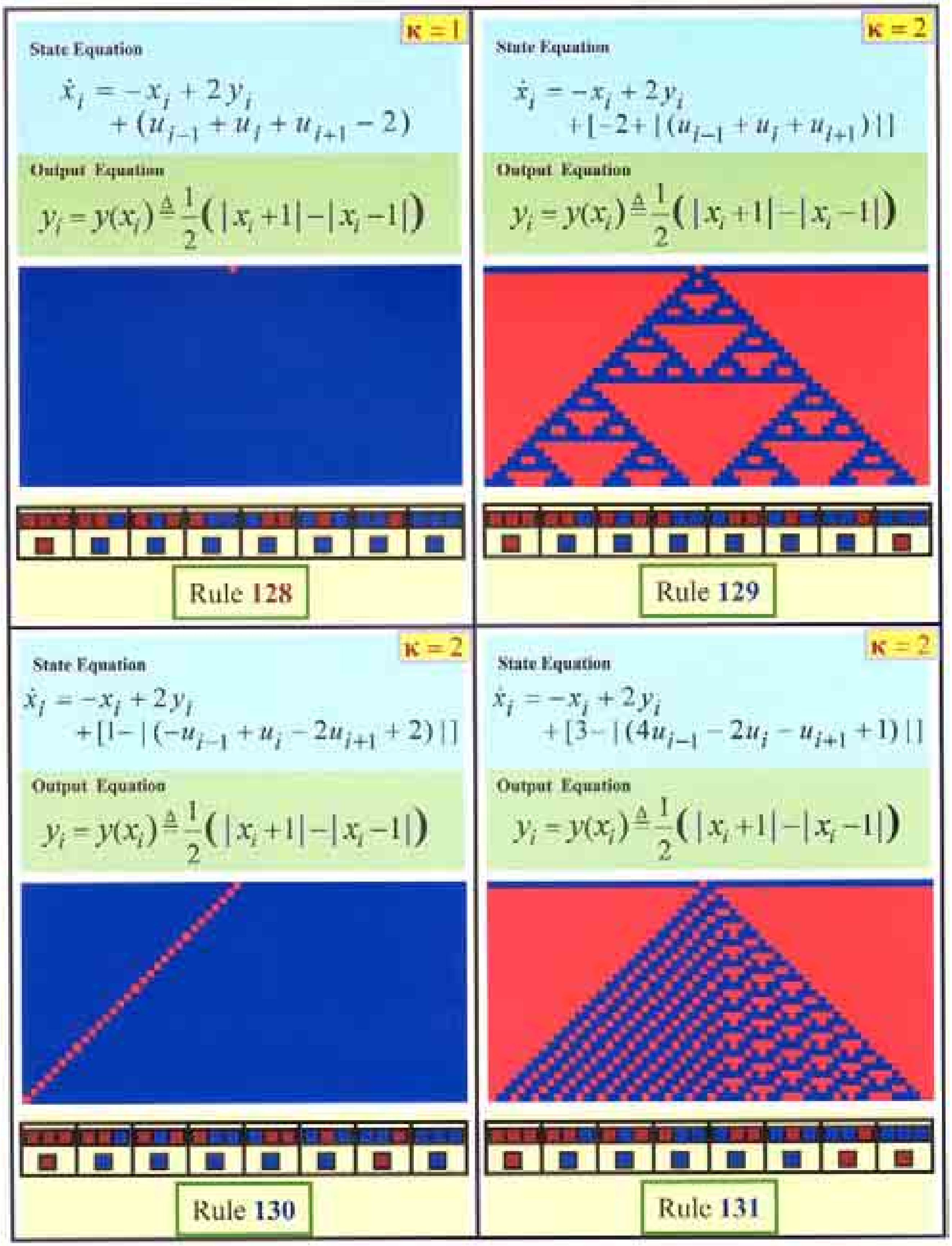


Table 2. (*Continued*)

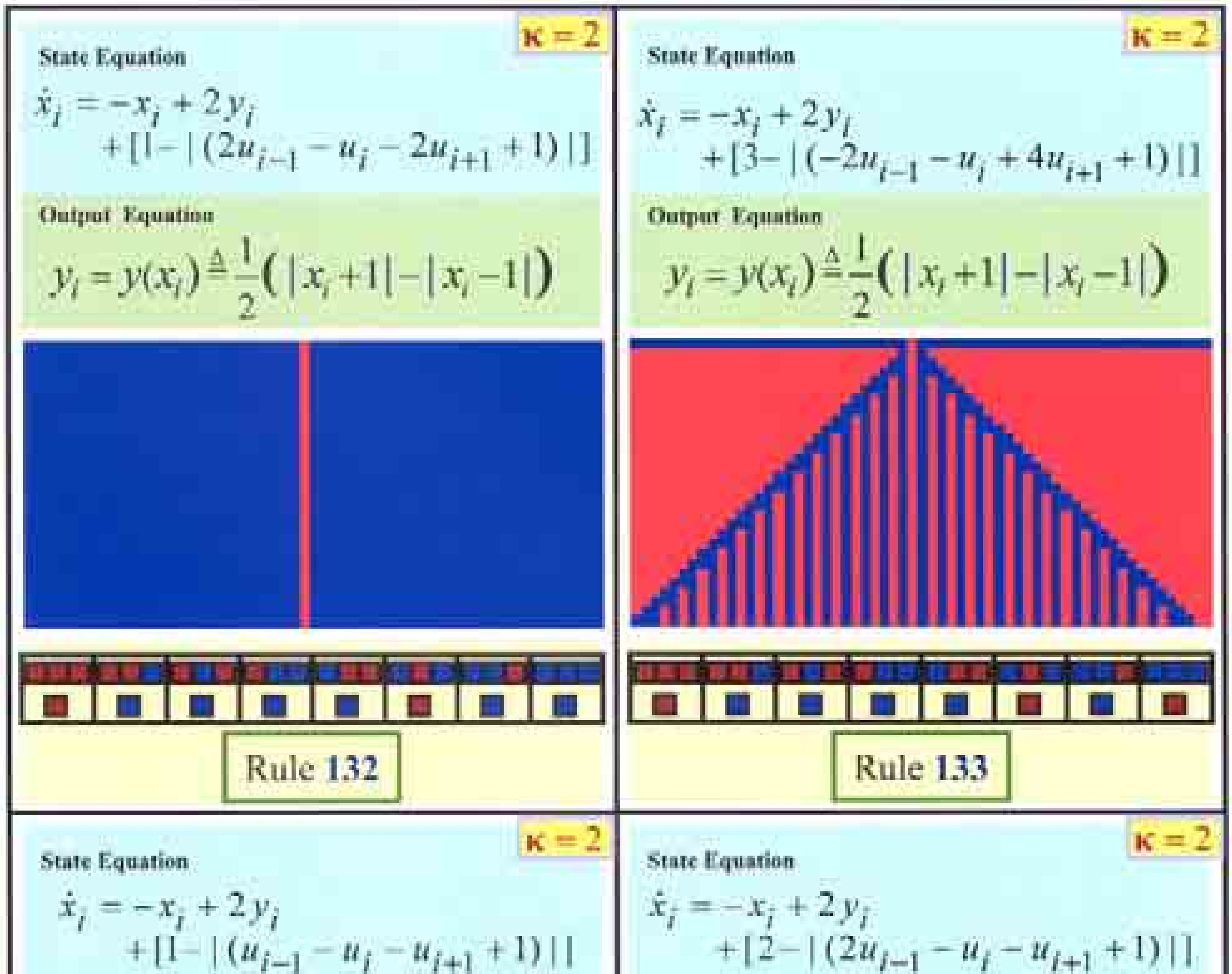


Table 2. (*Continued*)

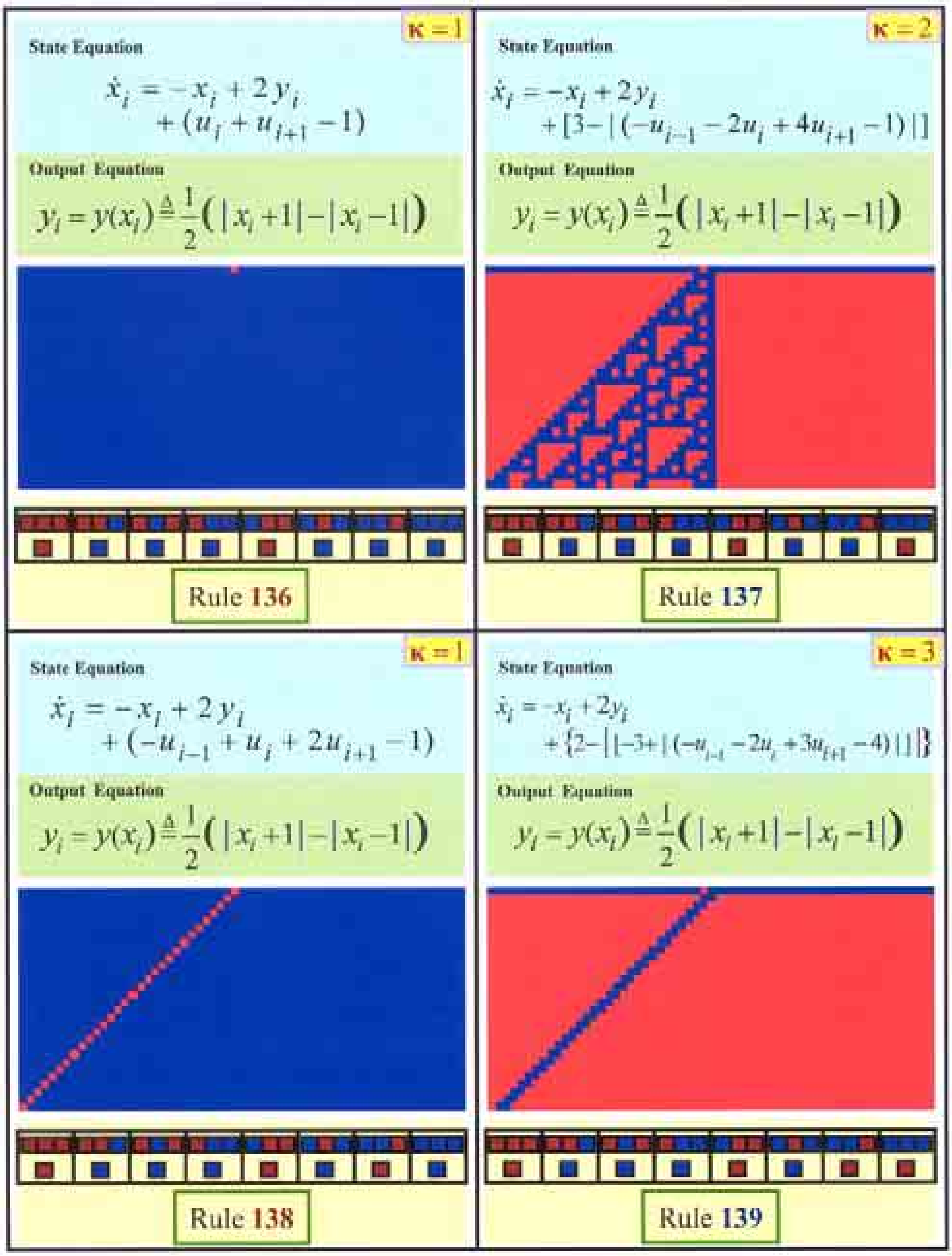


Table 2. (*Continued*)

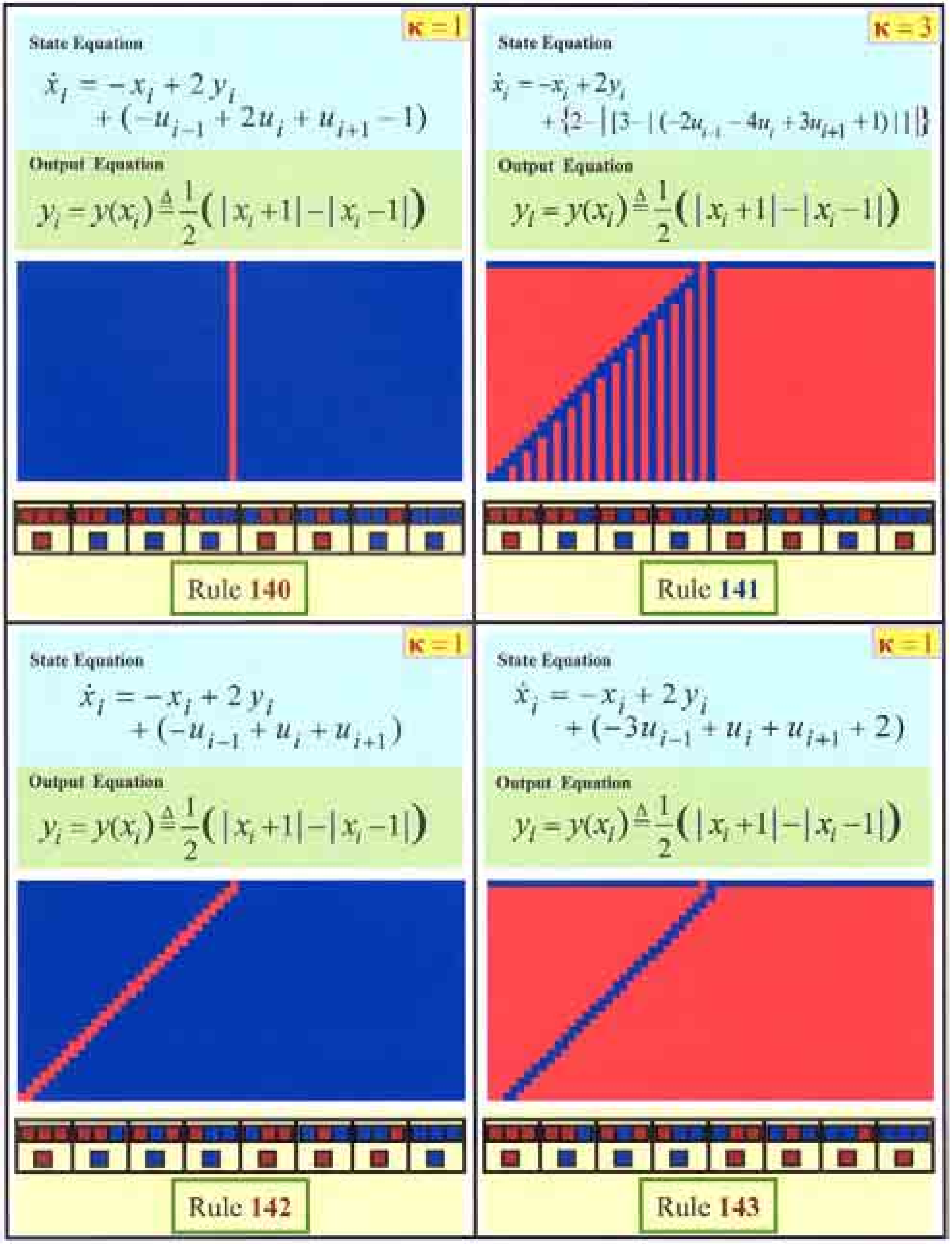


Table 2. (*Continued*)

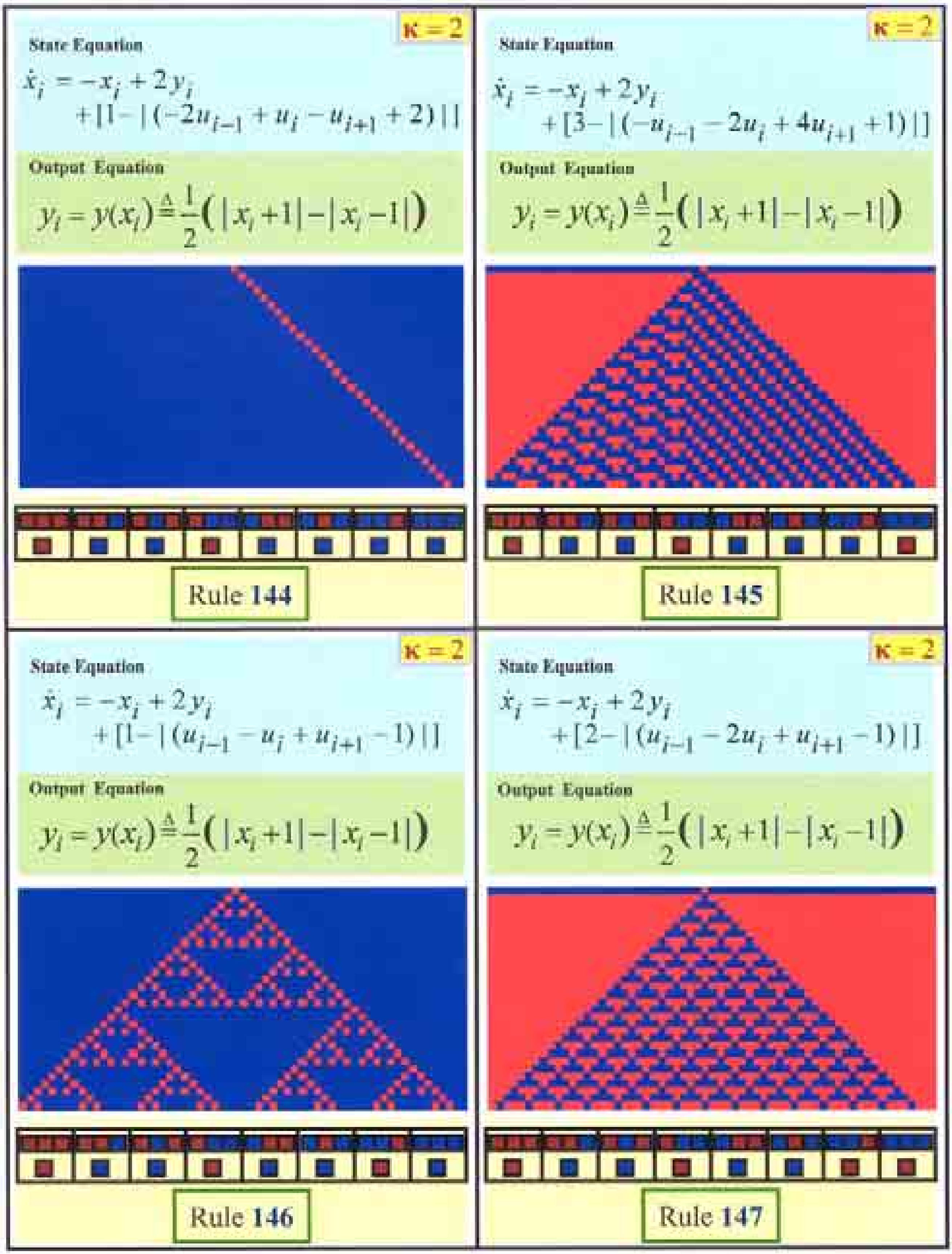


Table 2. (*Continued*)

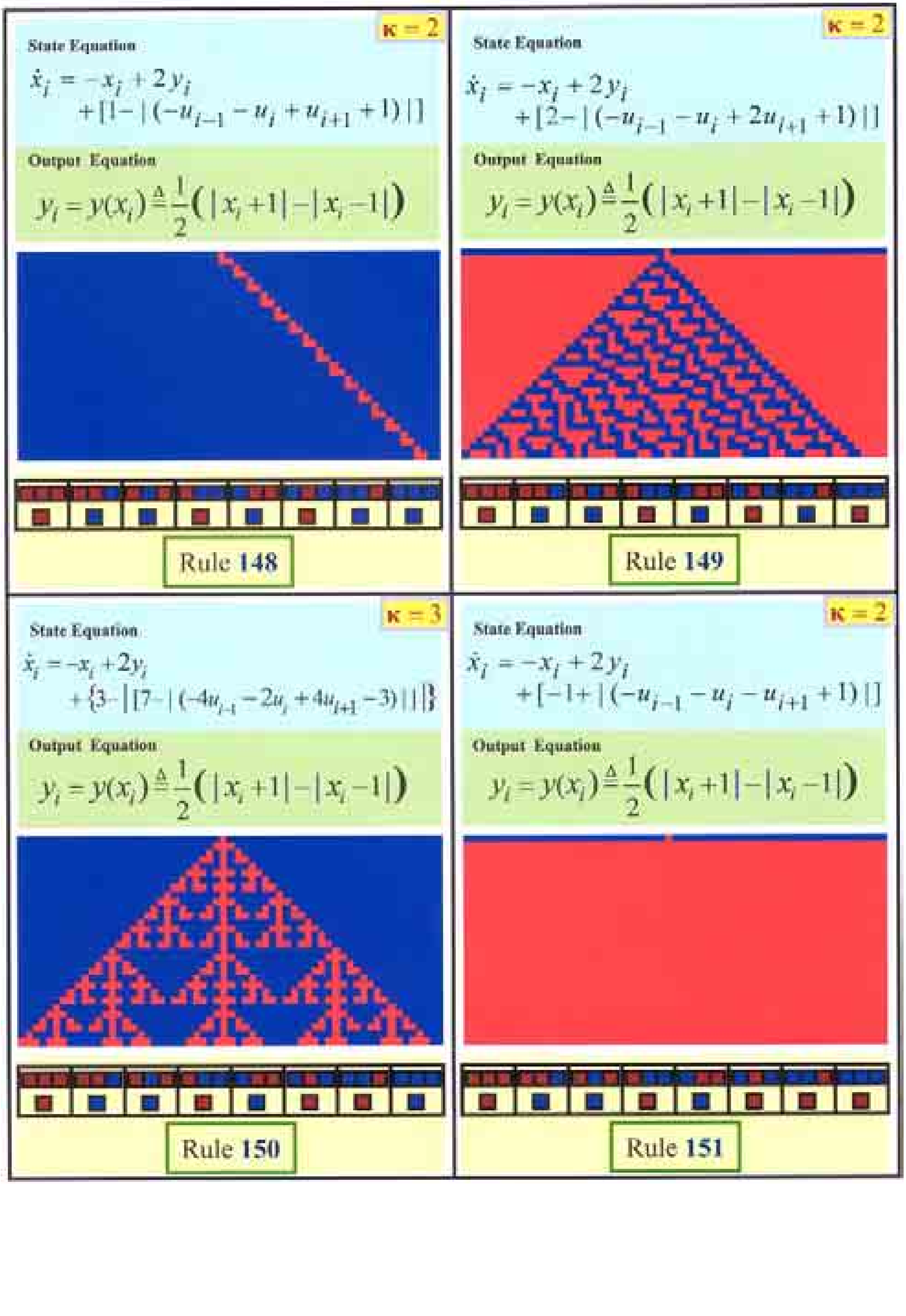


Table 2. (*Continued*)

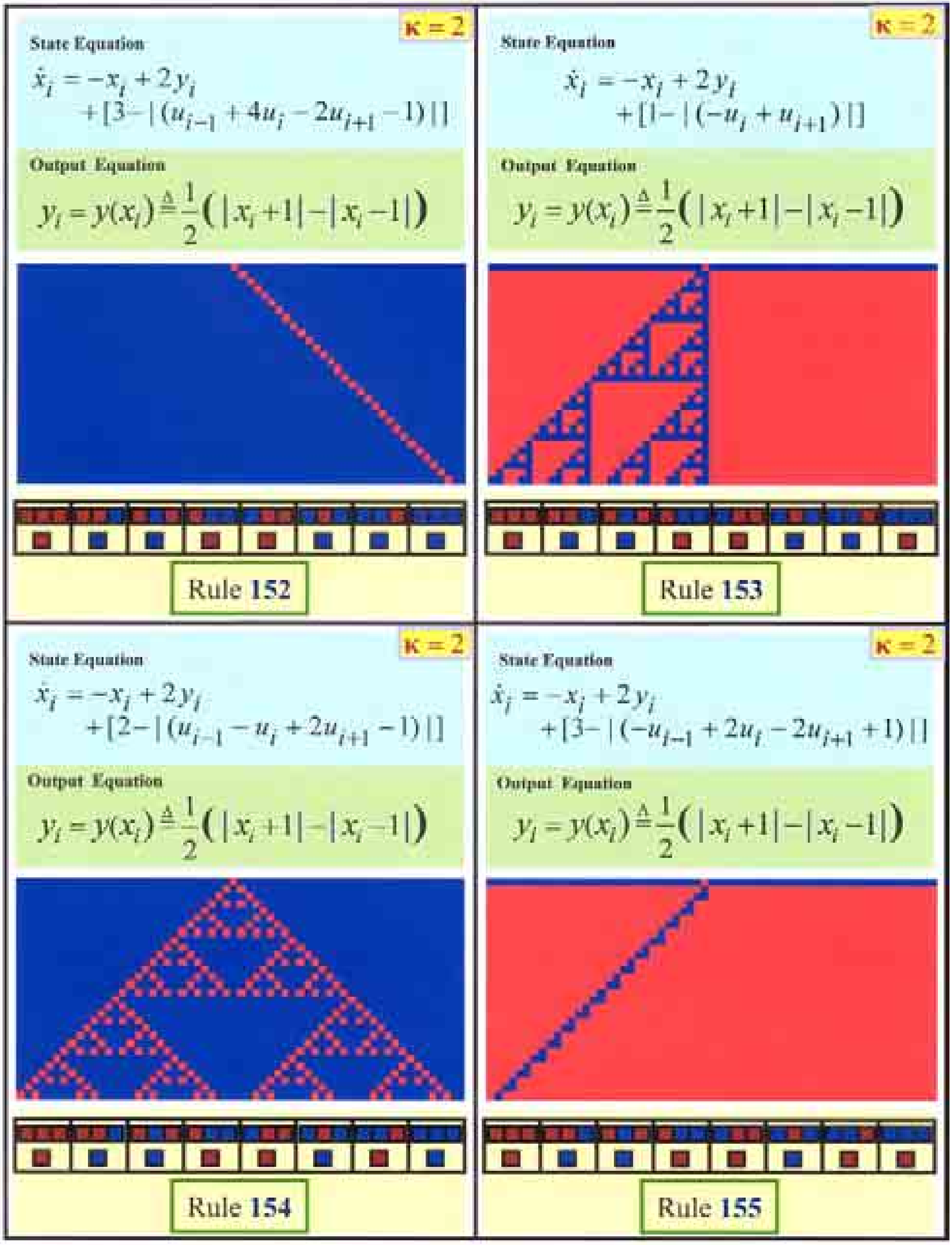


Table 2. (*Continued*)

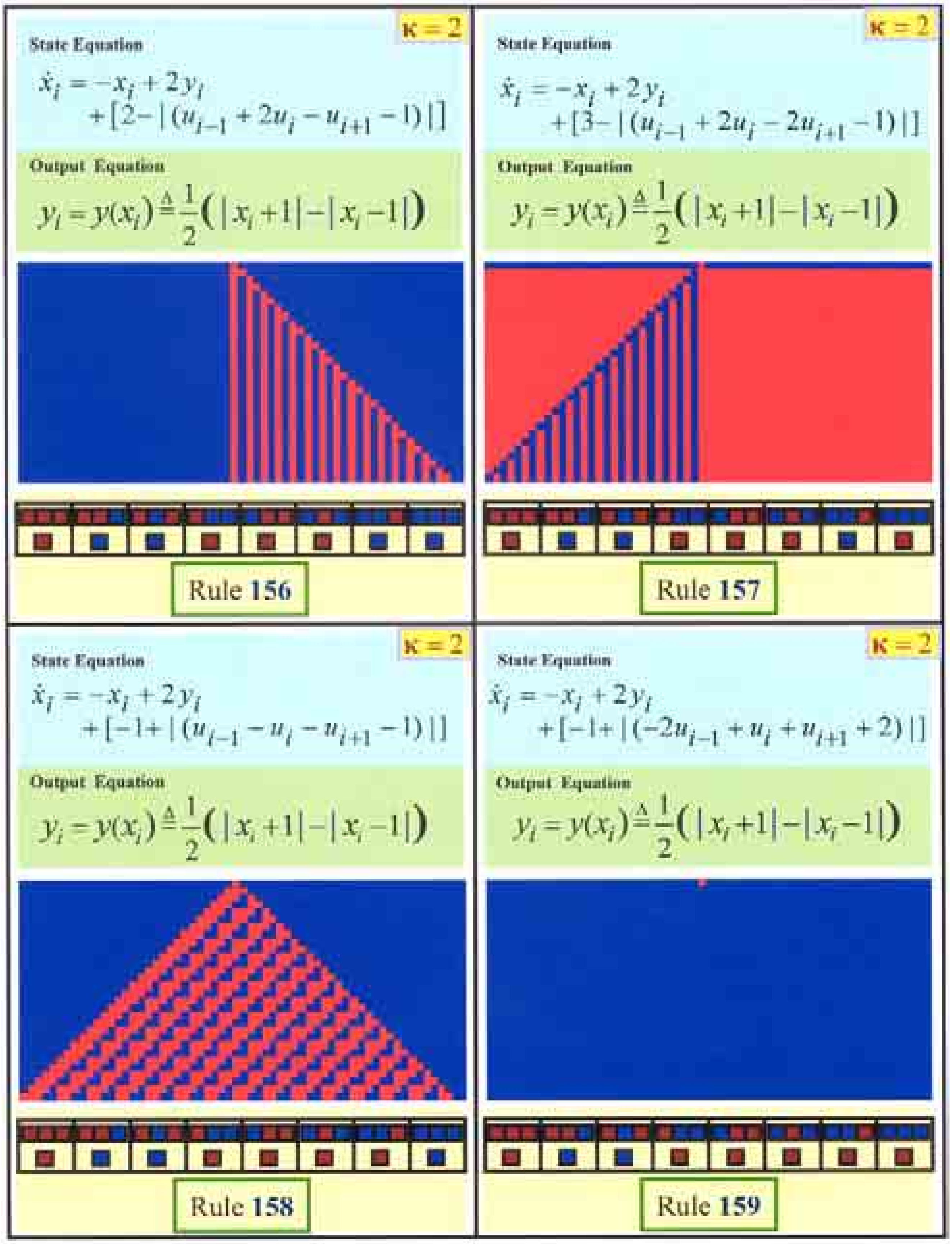


Table 2. (*Continued*)

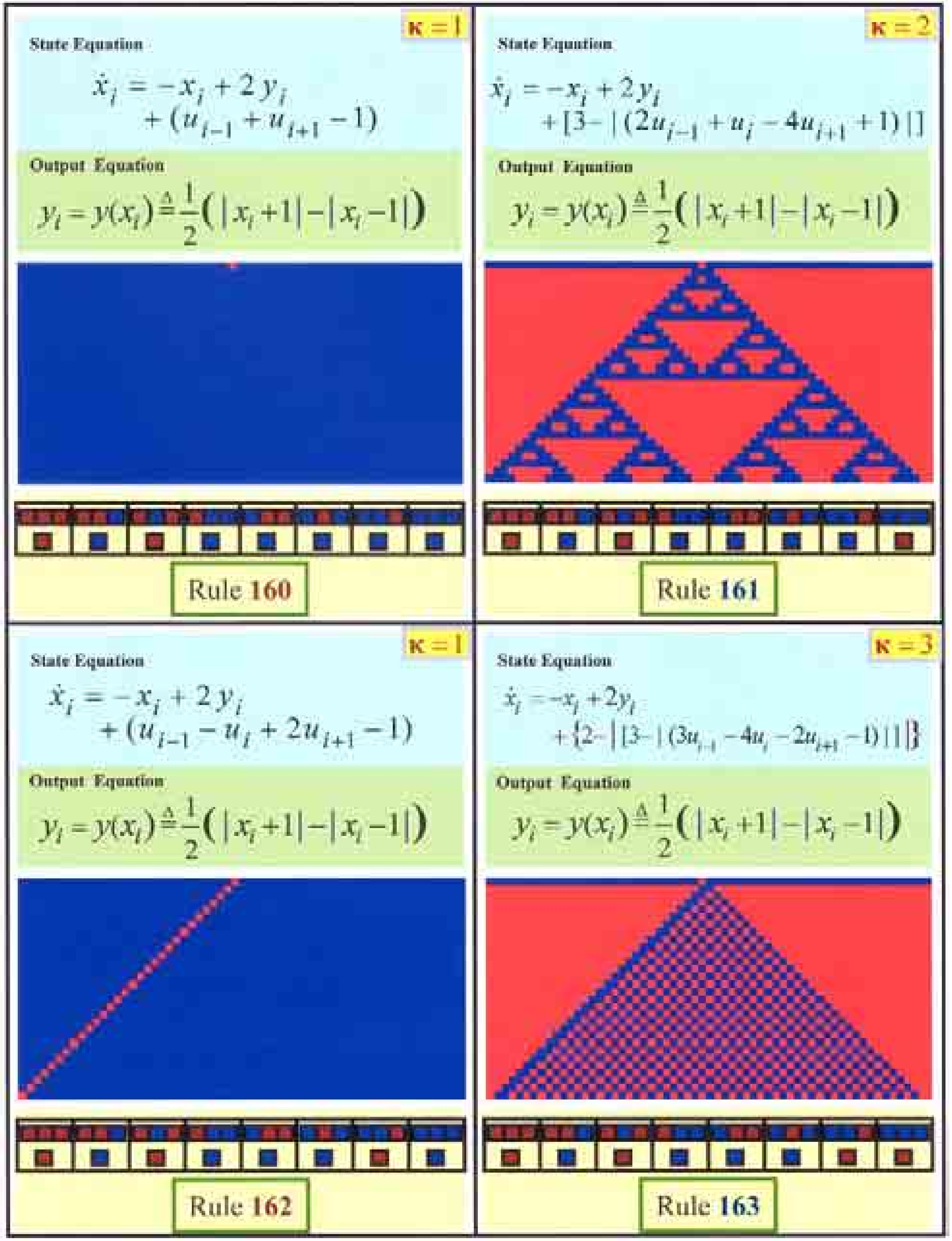


Table 2. (*Continued*)

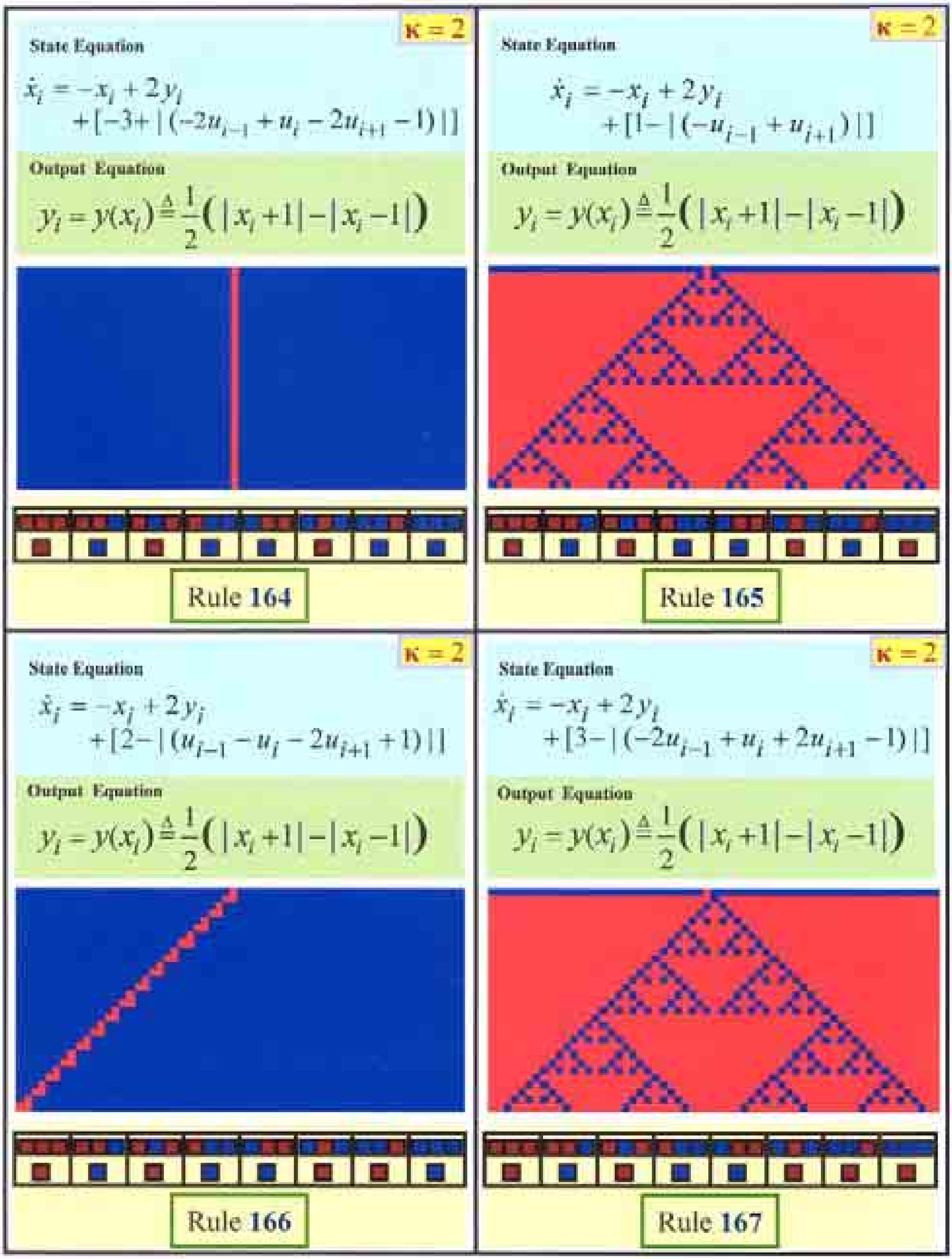


Table 2. (*Continued*)

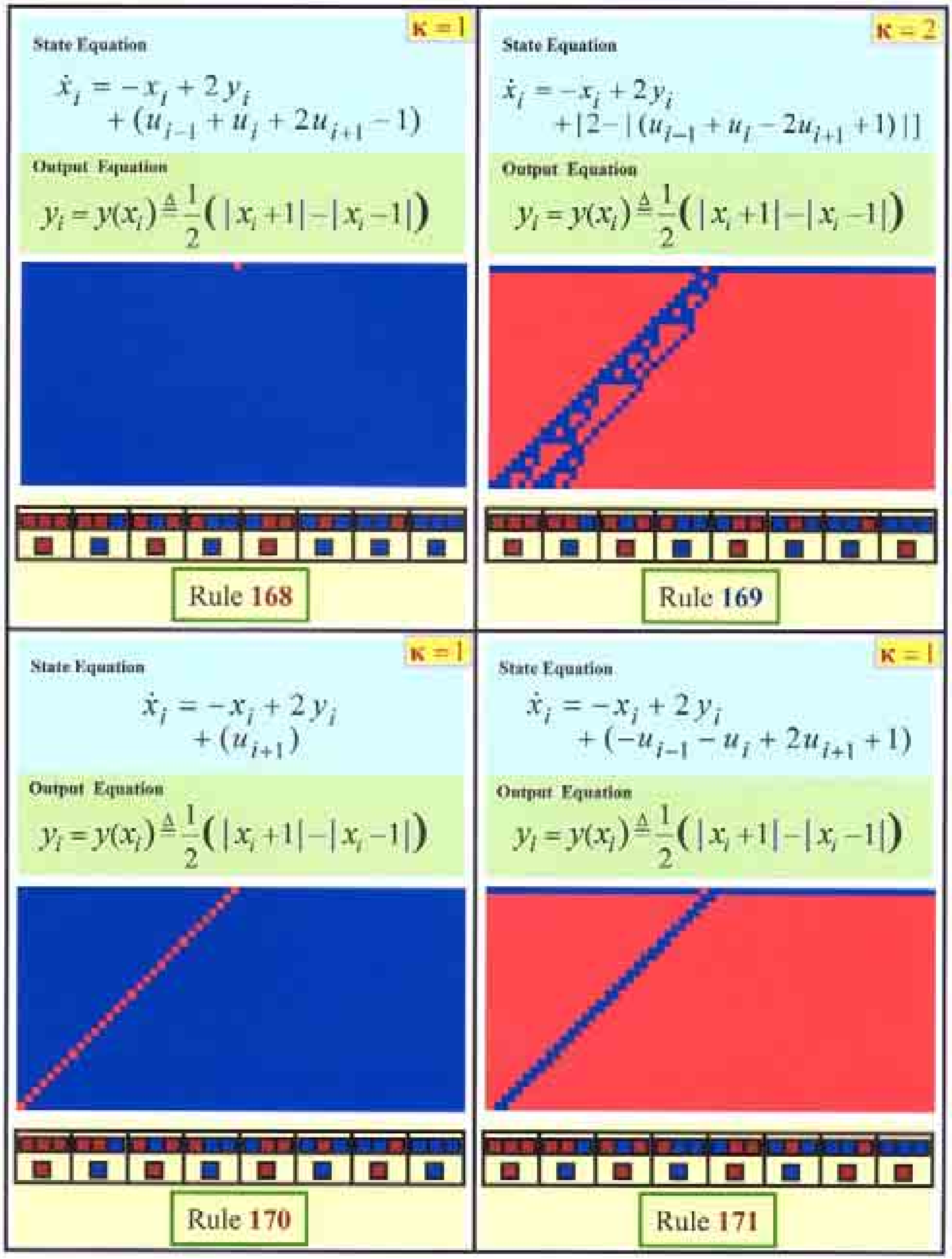


Table 2. (*Continued*)

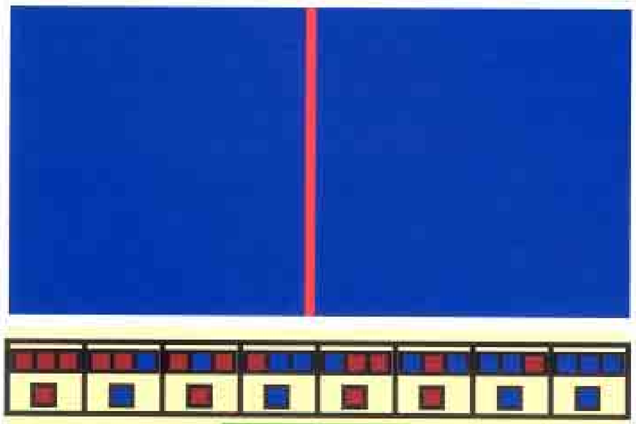
$\kappa = 3$

State Equation

$$\dot{x}_i = -x_i + 2y_i + \{2 - |[3 - |(-3u_{i-1} - 4u_i + 2u_{i+1} + 1)|]|\}$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **172**

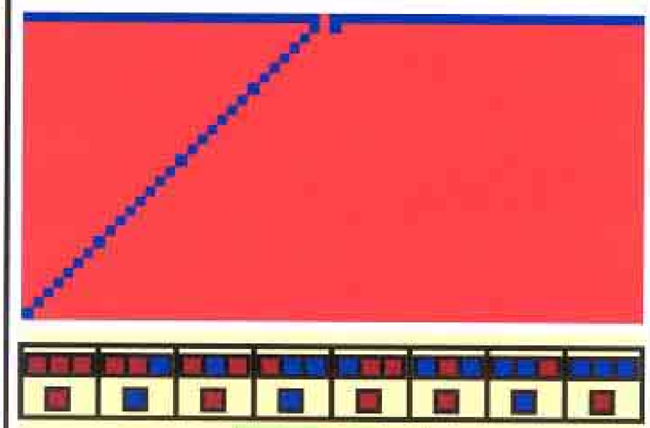
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(-2u_{i-1} - u_i + 2u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **173**

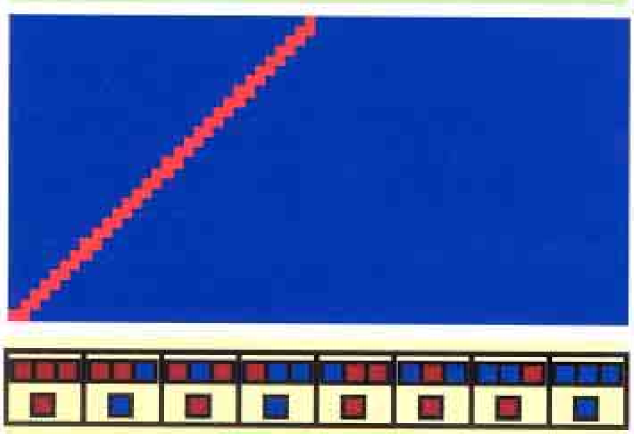
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} + u_i + 2u_{i+1} + 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **174**

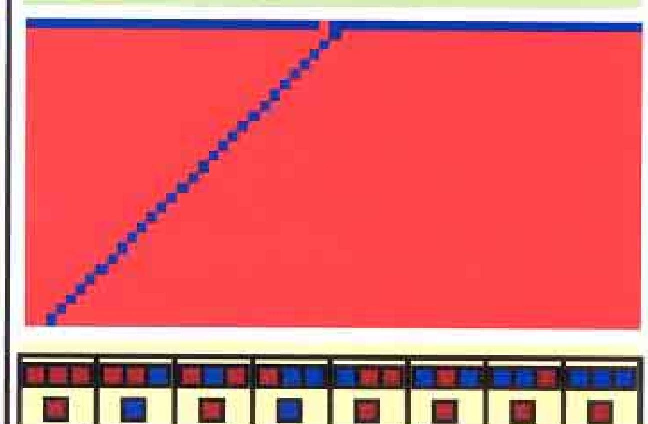
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} + u_{i+1} + 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **175**

Table 2. (*Continued*)

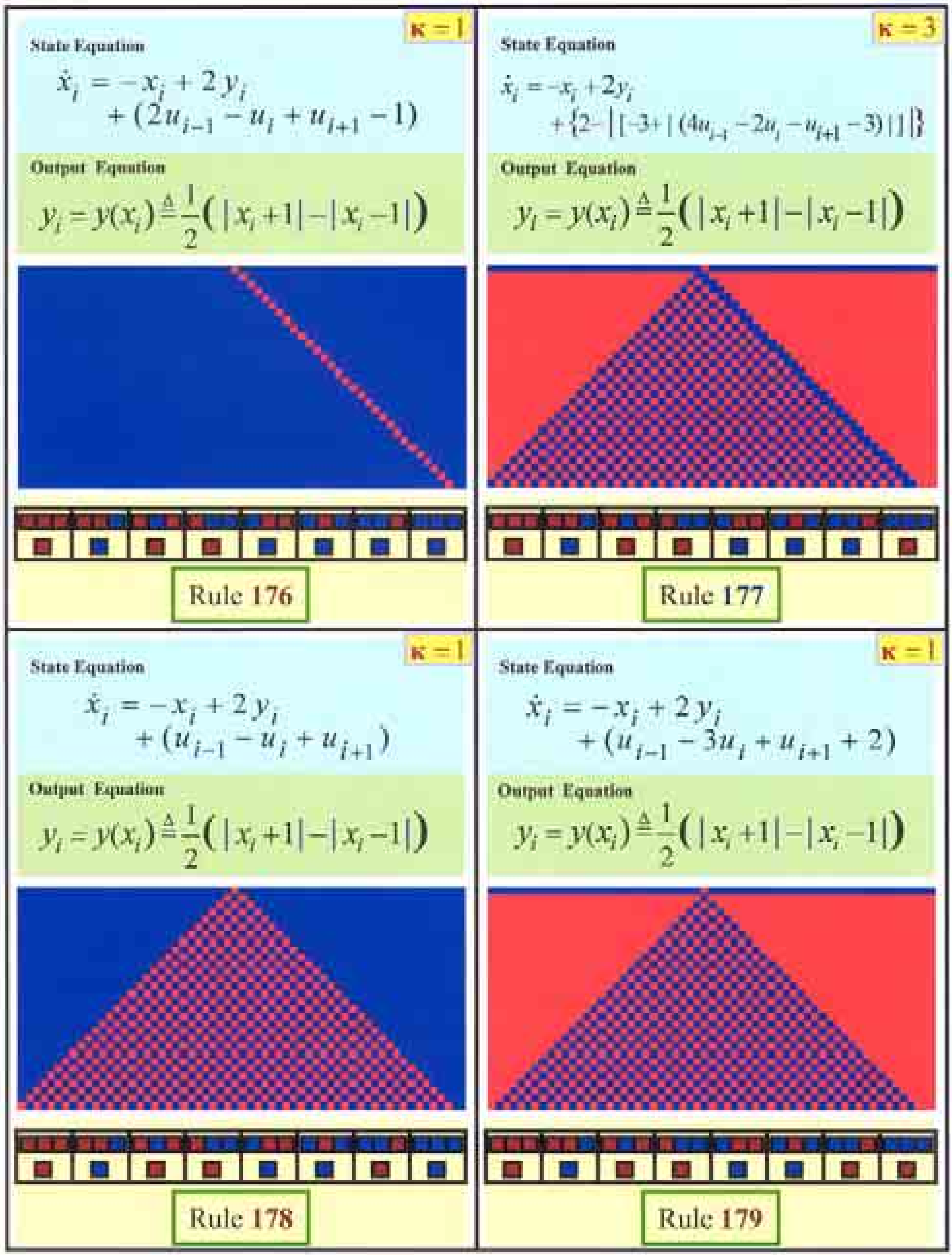


Table 2. (*Continued*)

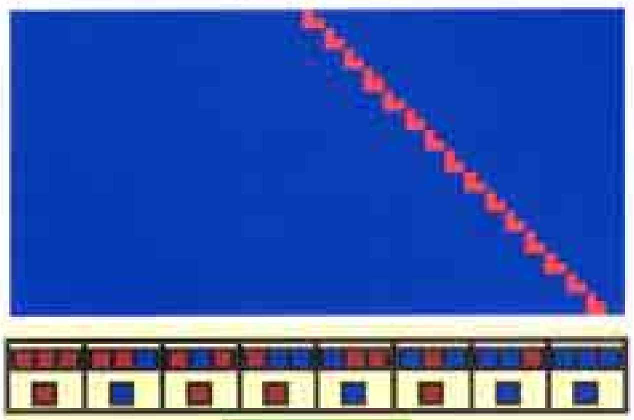
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [2 - |(-2u_{i-1} - u_i + u_{i+1} + 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 180

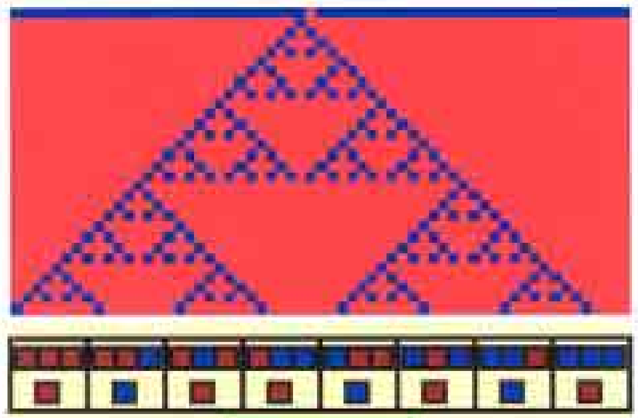
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [3 - |(2u_{i-1} + u_i - 2u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule 181

$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [-1 + |(-u_{i-1} + u_i - u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$

Rule 182

$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [-1 + |(u_{i-1} - 2u_i + u_{i+1} + 2)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$

Rule 183

Table 2. (*Continued*)

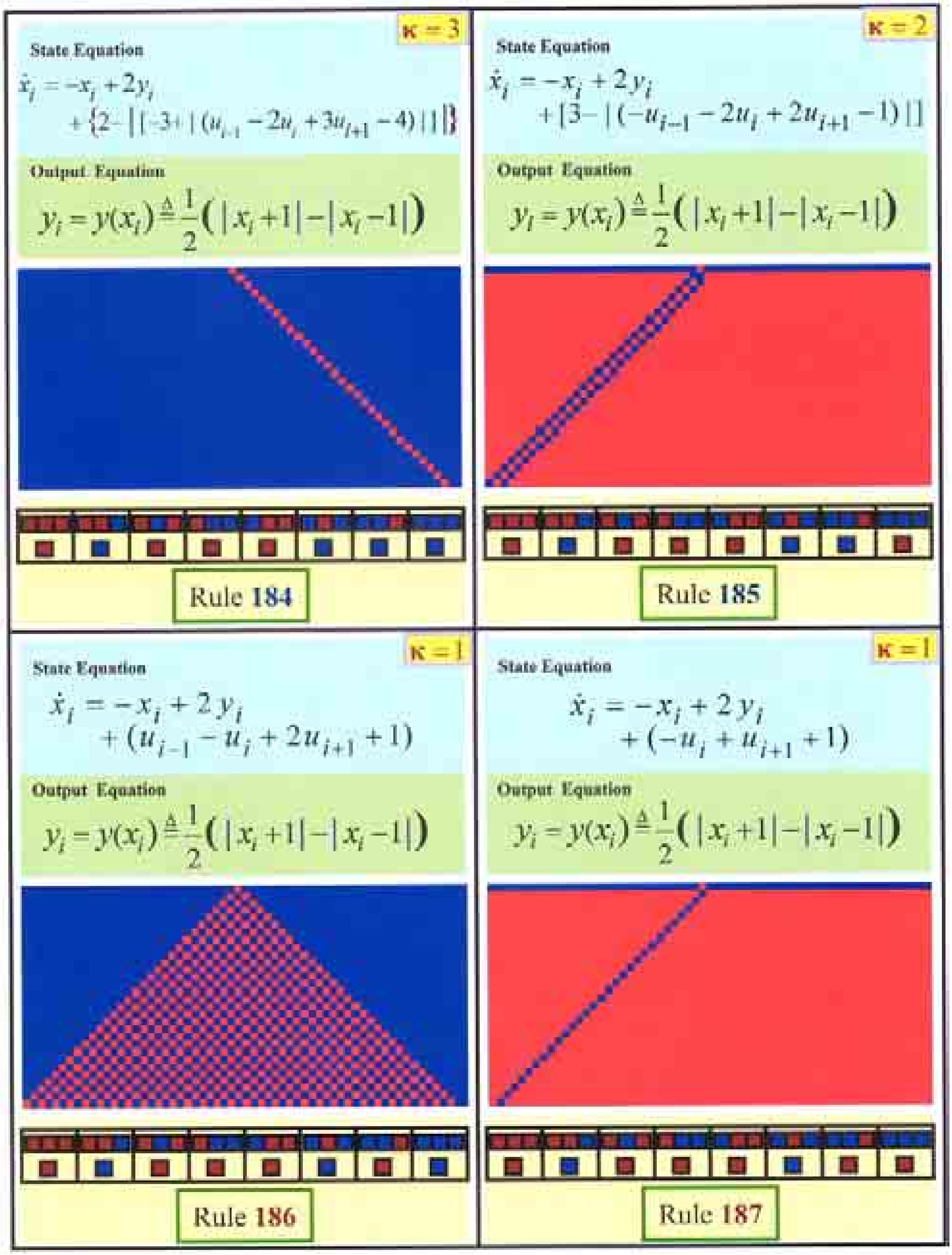


Table 2. (*Continued*)

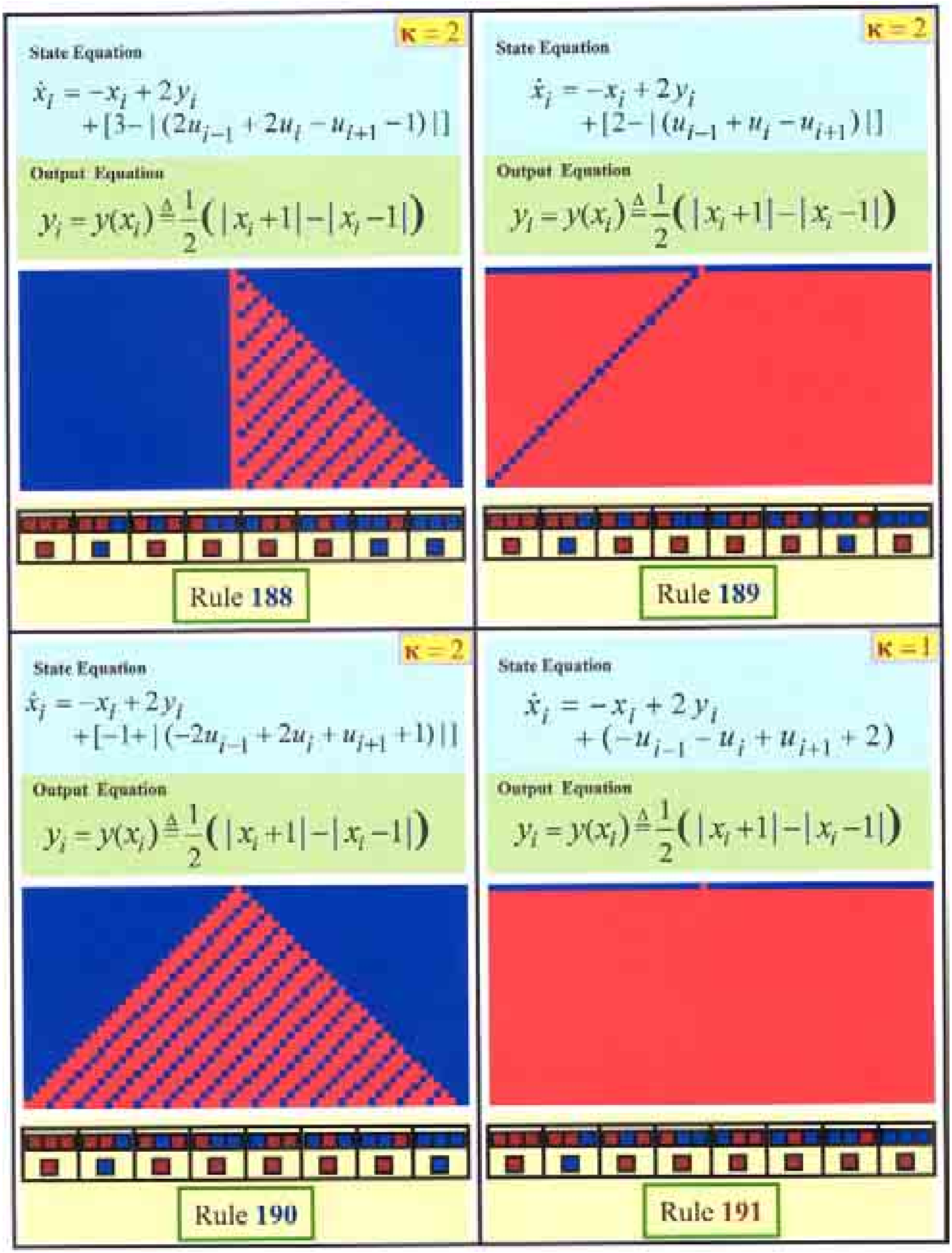


Table 2. (*Continued*)

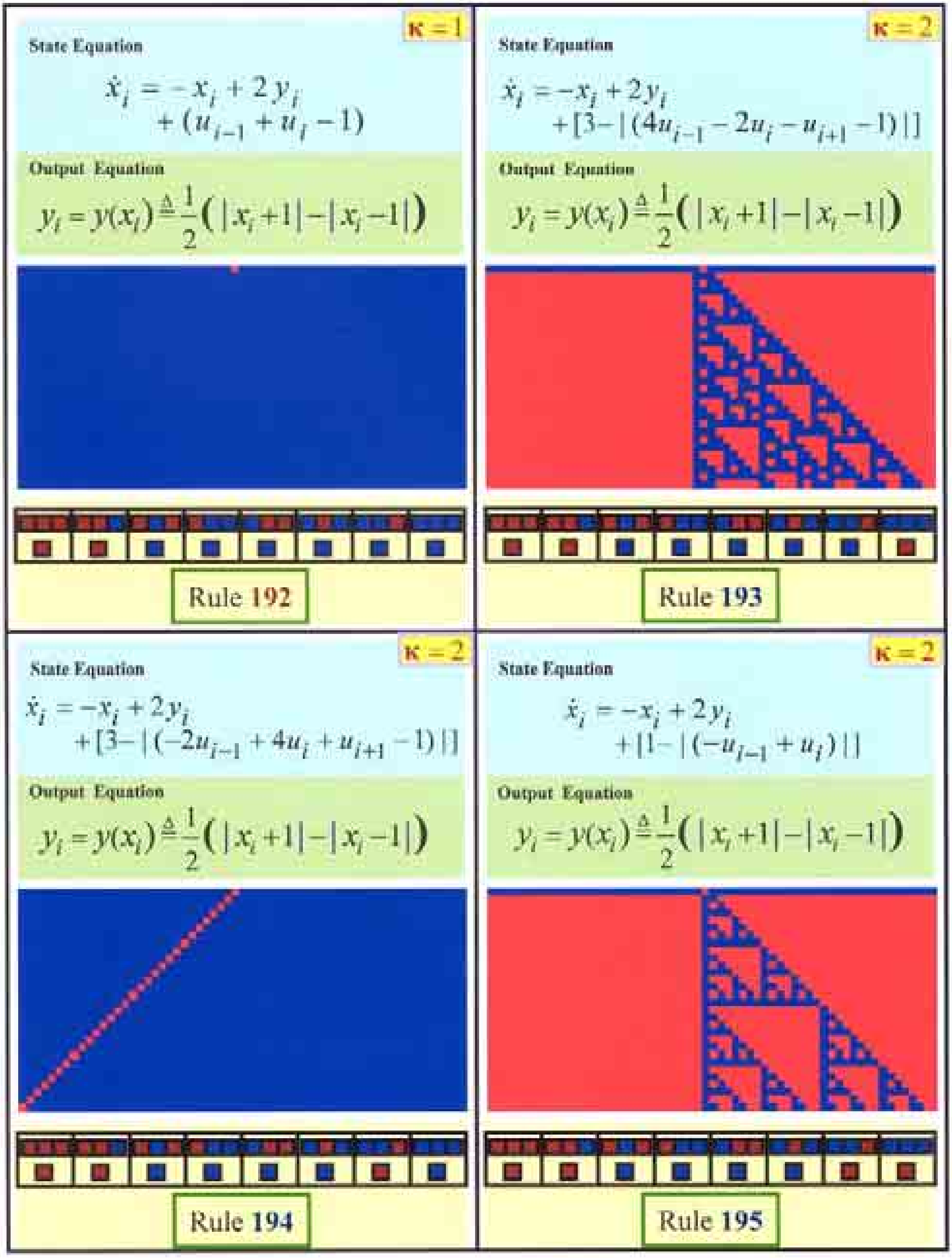


Table 2. (*Continued*)

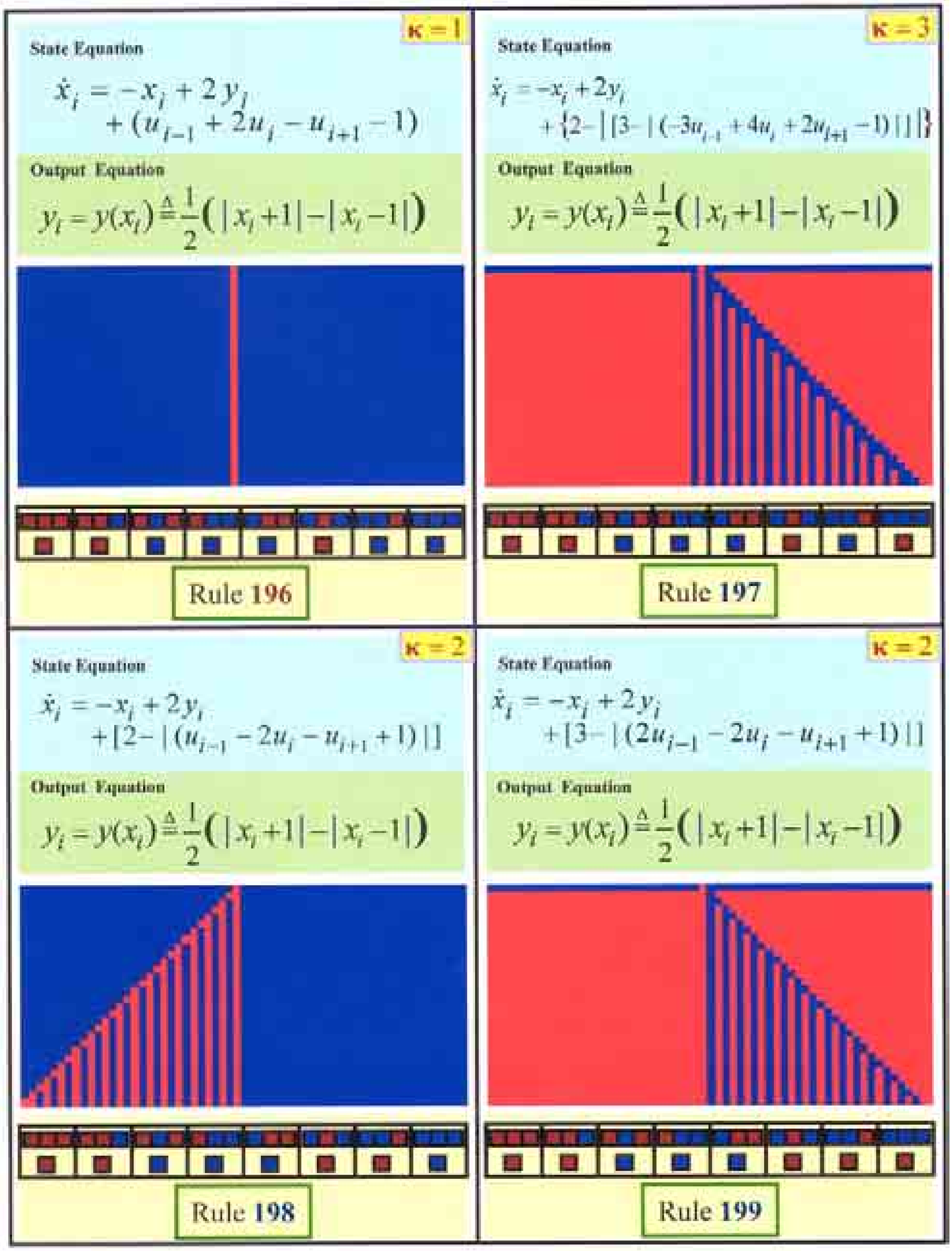


Table 2. (*Continued*)

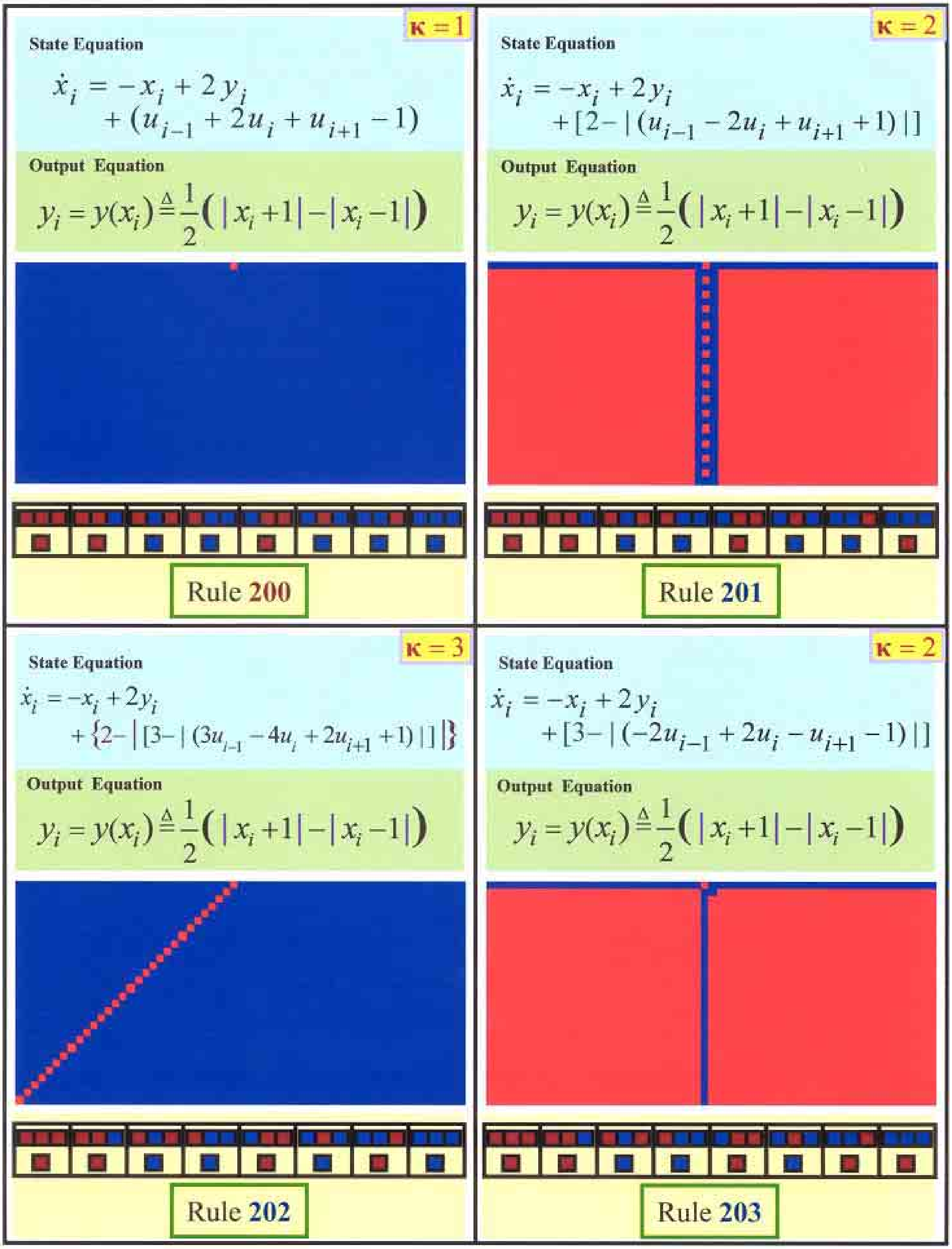


Table 2. (*Continued*)

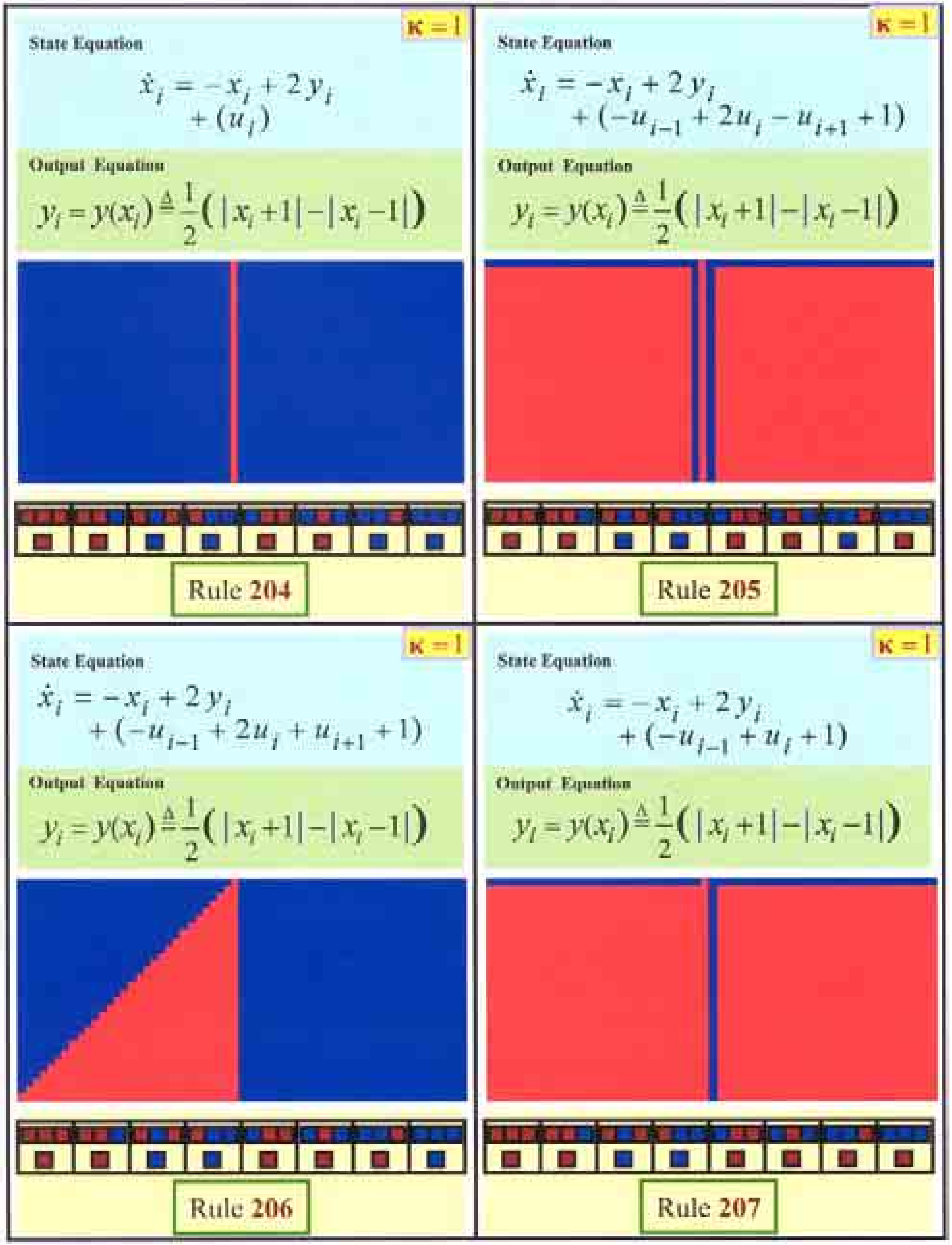


Table 2. (*Continued*)

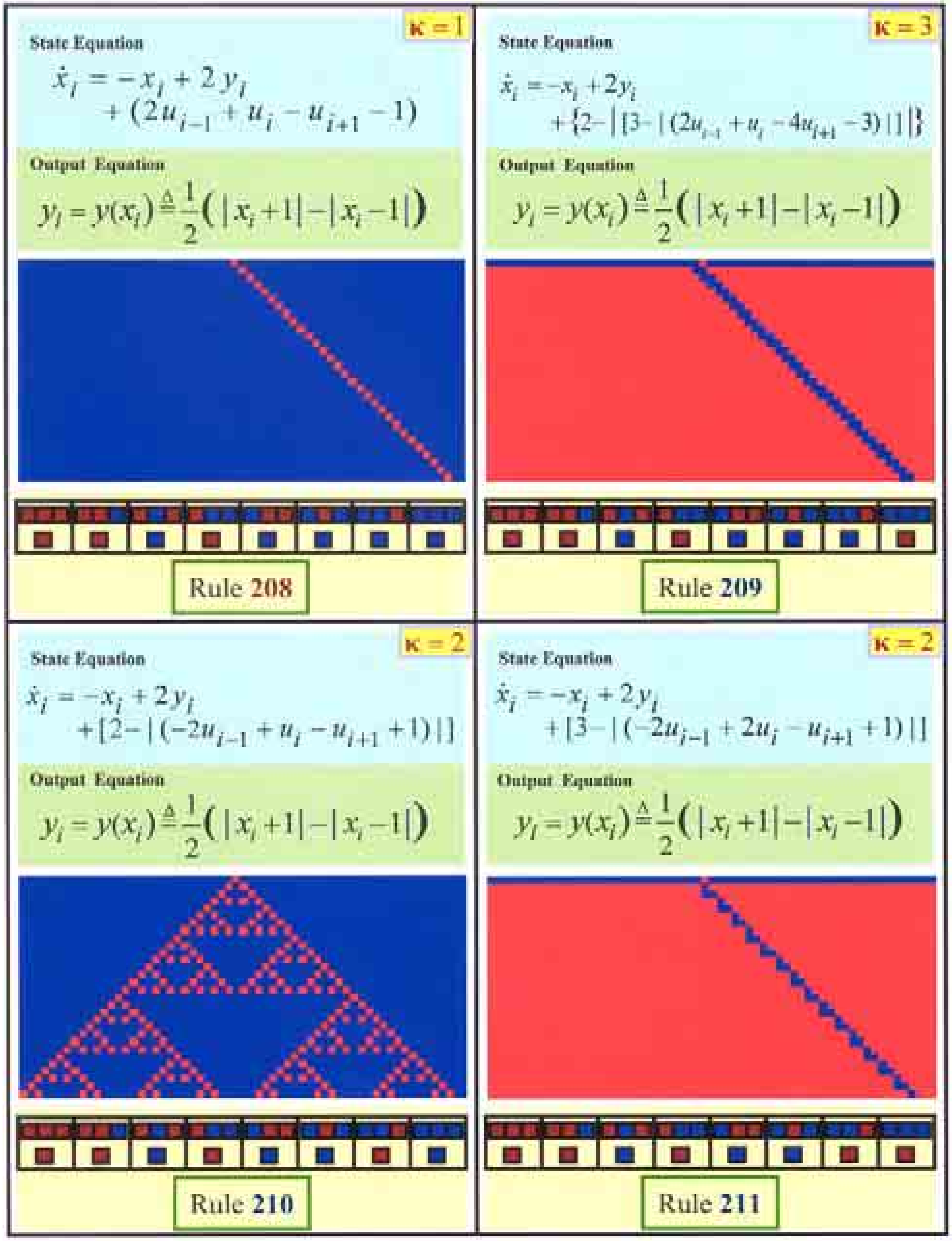


Table 2. (*Continued*)

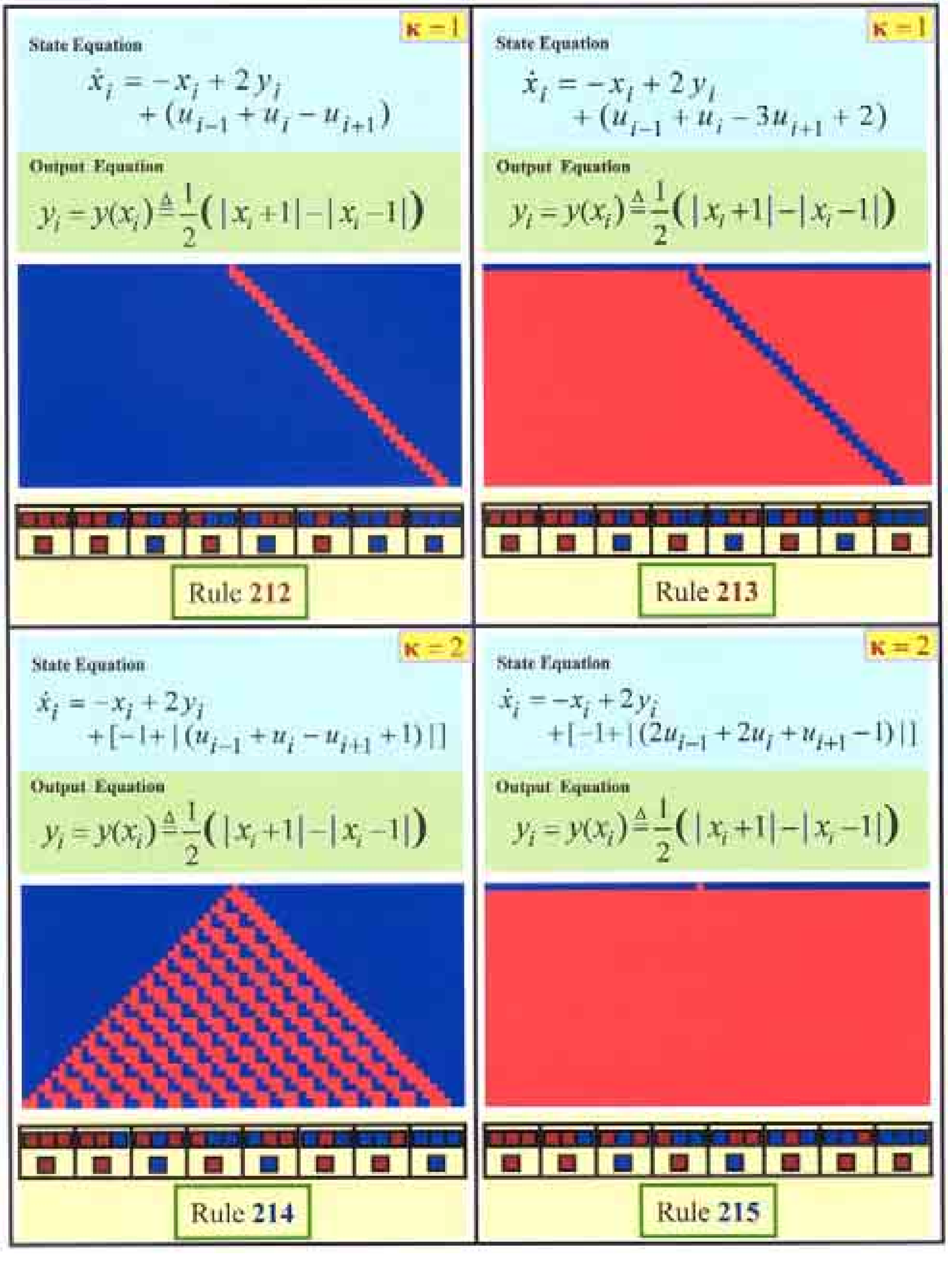


Table 2. (*Continued*)

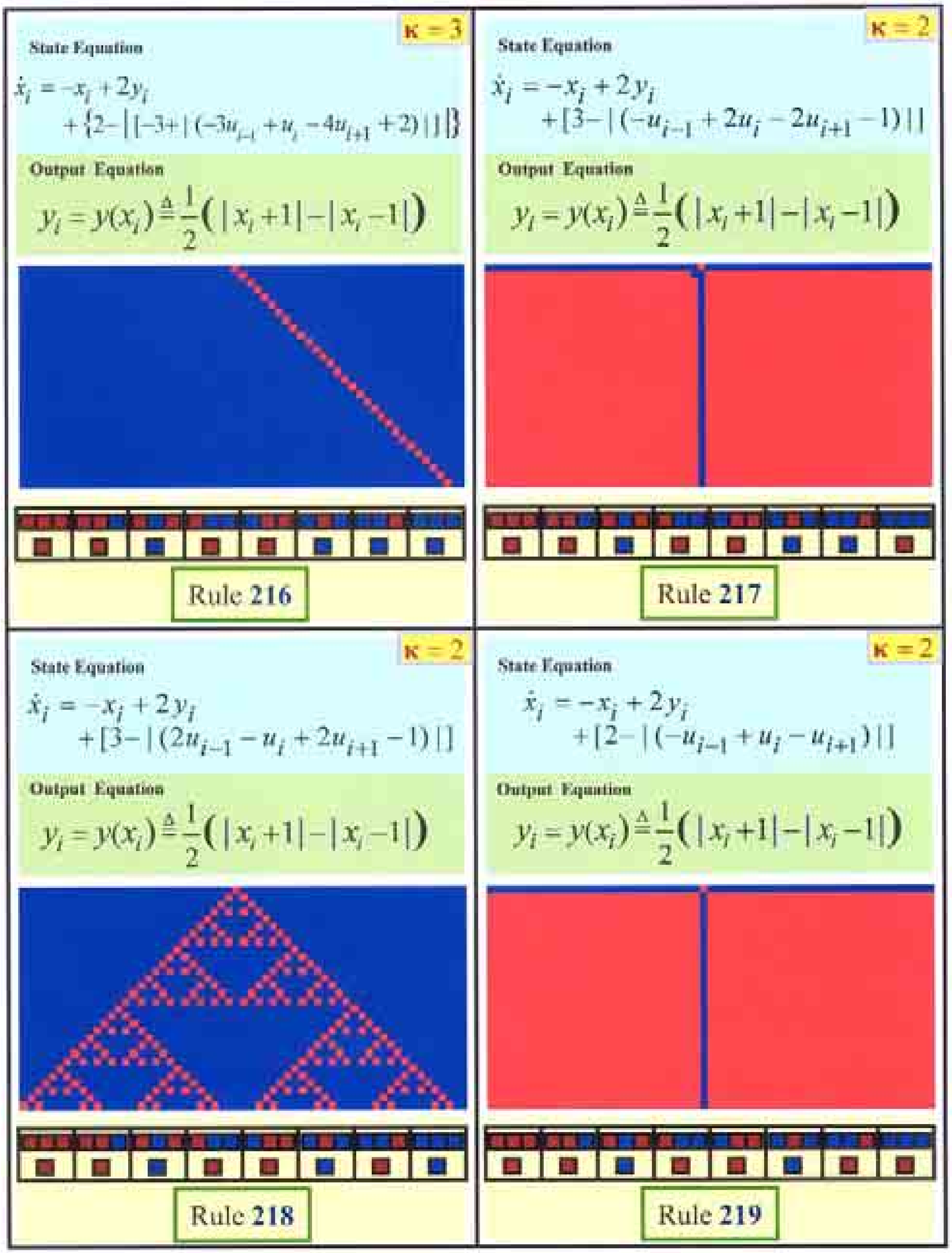


Table 2. (*Continued*)

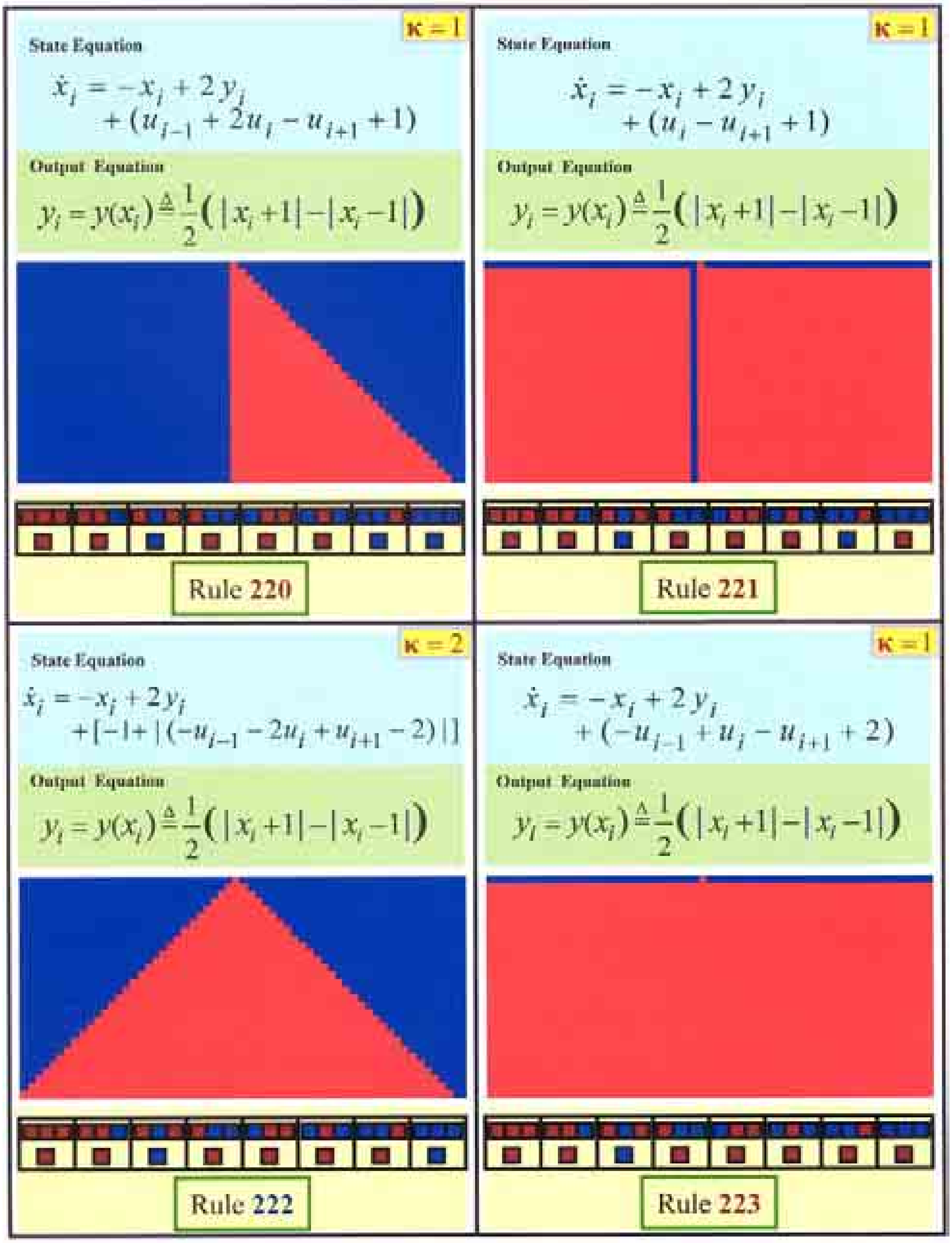


Table 2. (*Continued*)

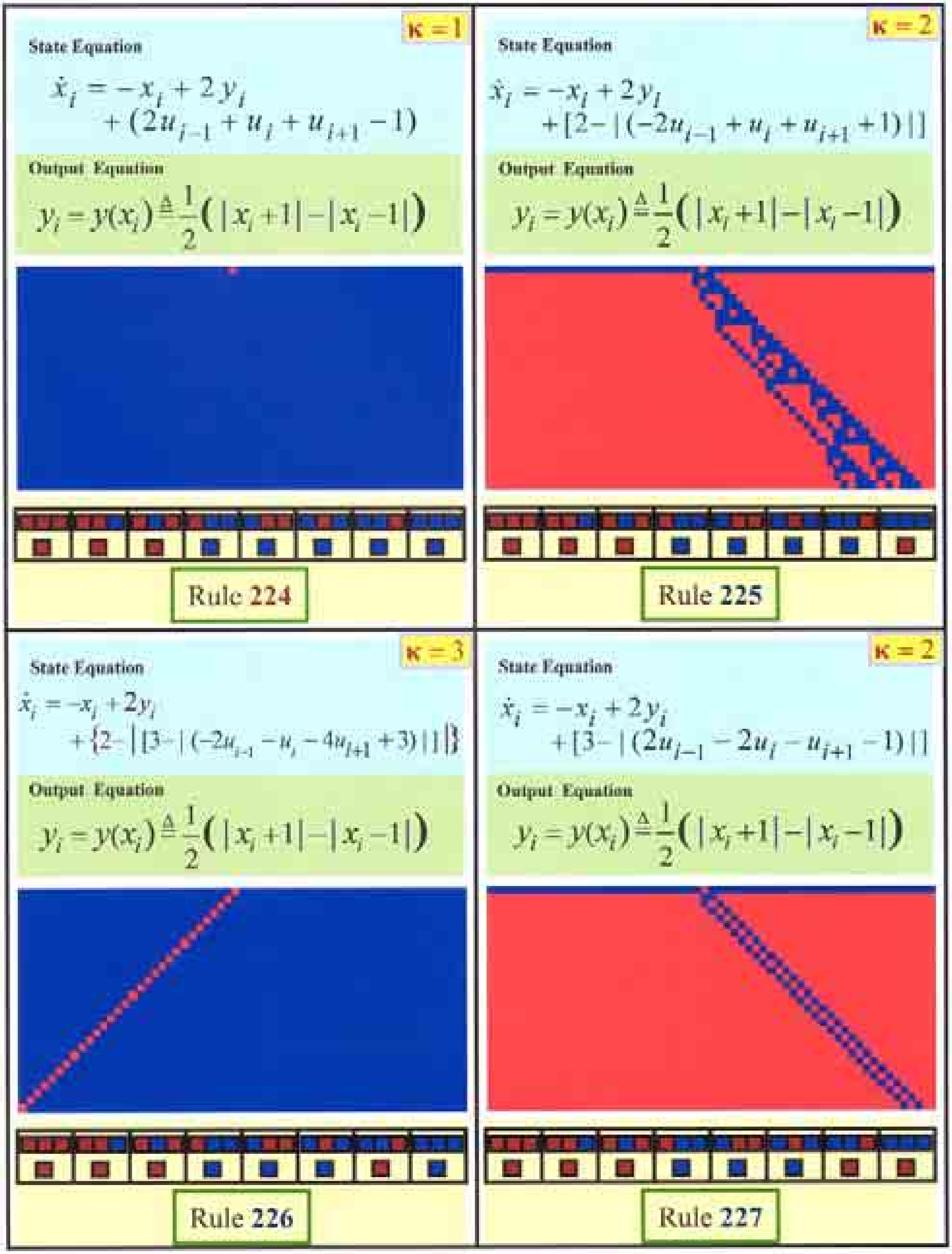


Table 2. (*Continued*)

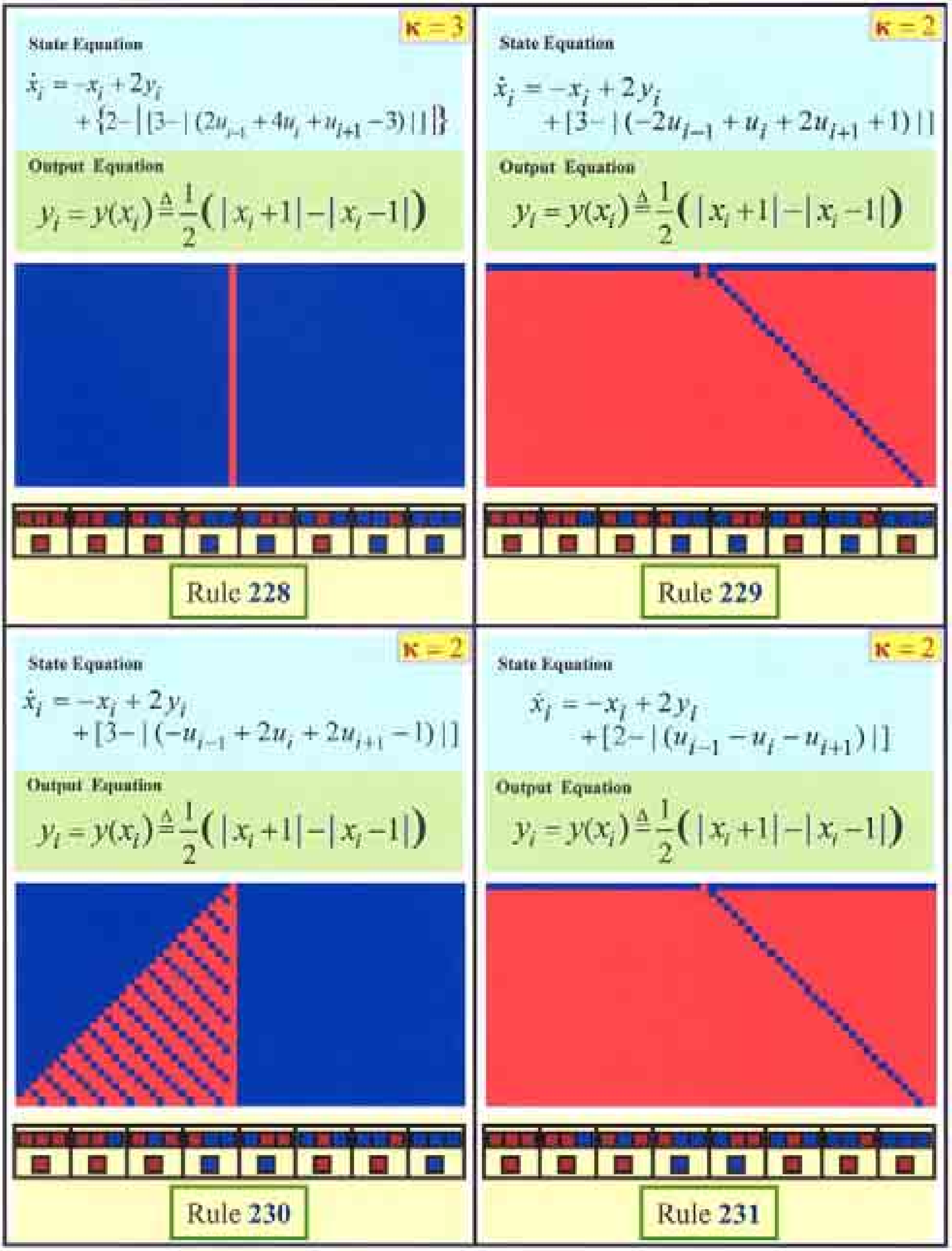


Table 2. (*Continued*)

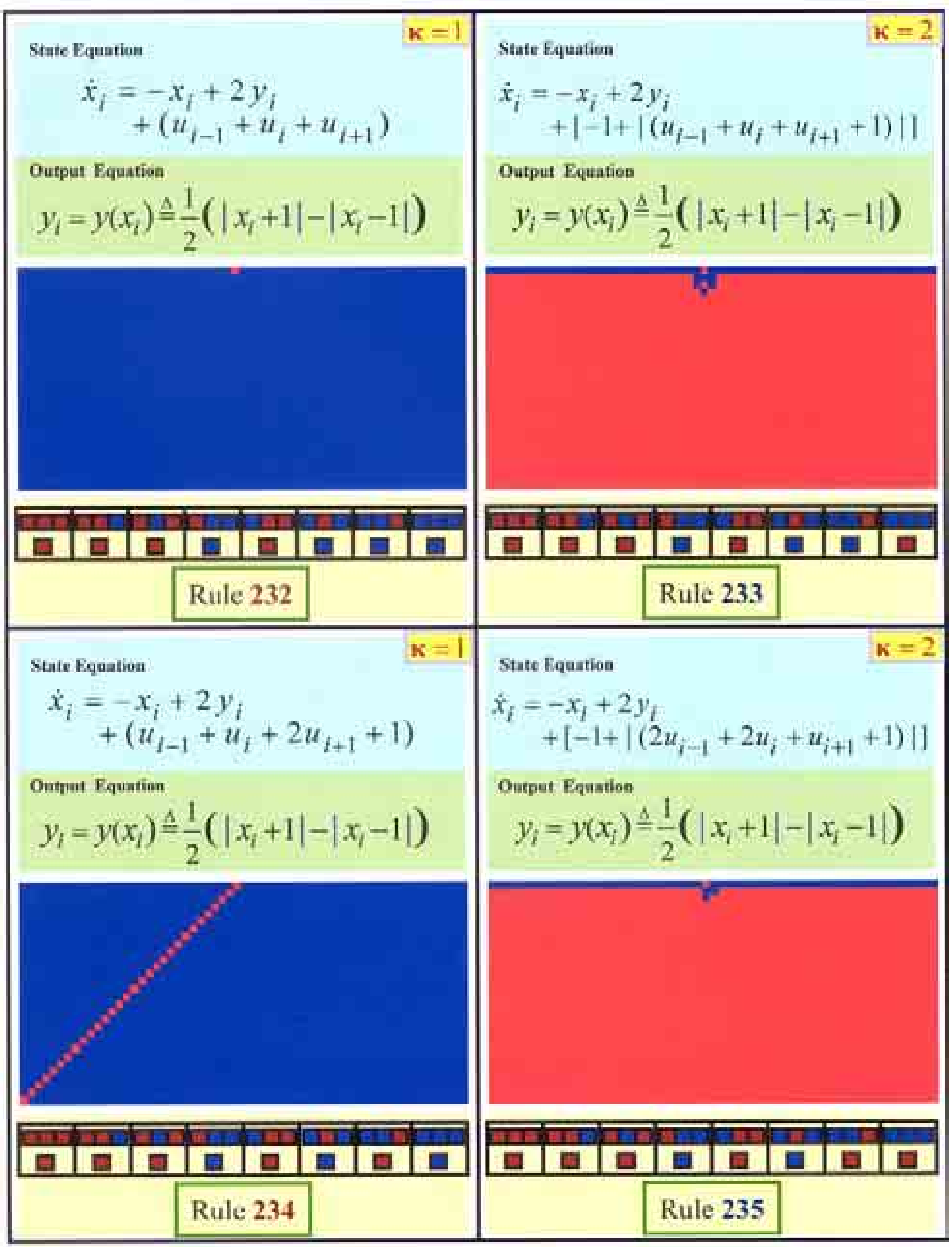


Table 2. (*Continued*)

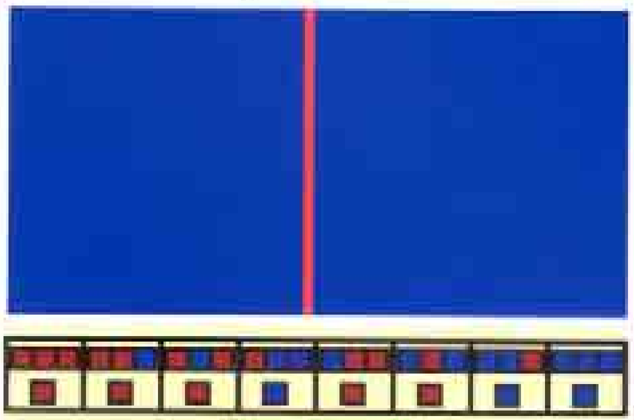
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (u_{i-1} + 2u_i + u_{i+1} + 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **236**

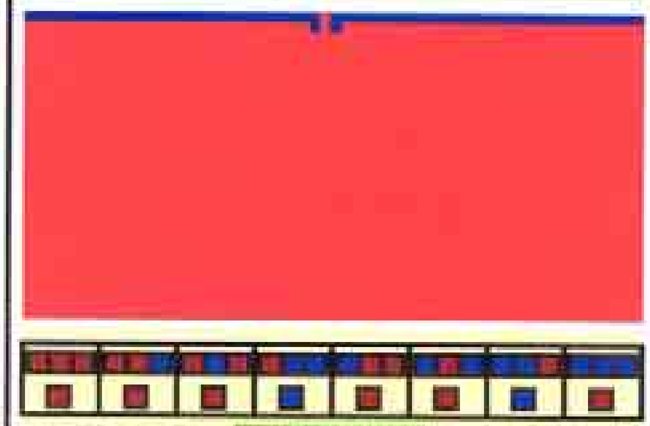
$\kappa = 2$

State Equation

$$\dot{x}_i = -x_i + 2y_i + [-1 + |(2u_{i-1} - u_i + 2u_{i+1} - 1)|]$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **237**

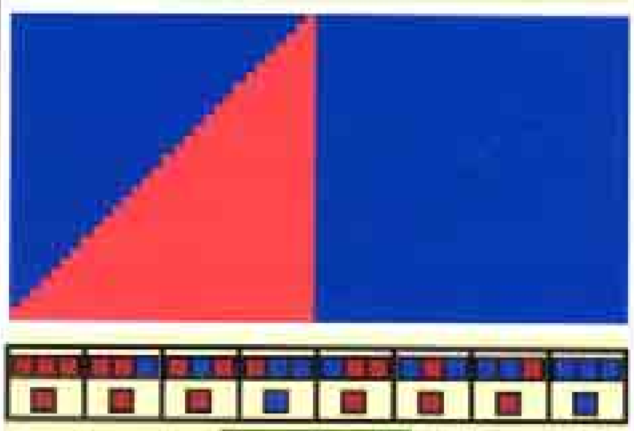
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (u_i + u_{i+1} + 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **238**

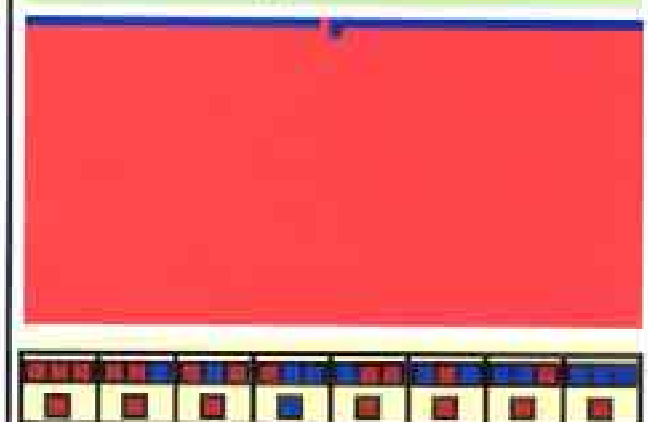
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (-u_{i-1} + u_i + u_{i+1} + 2)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i + 1| - |x_i - 1|)$$



Rule **239**

Table 2. (*Continued*)

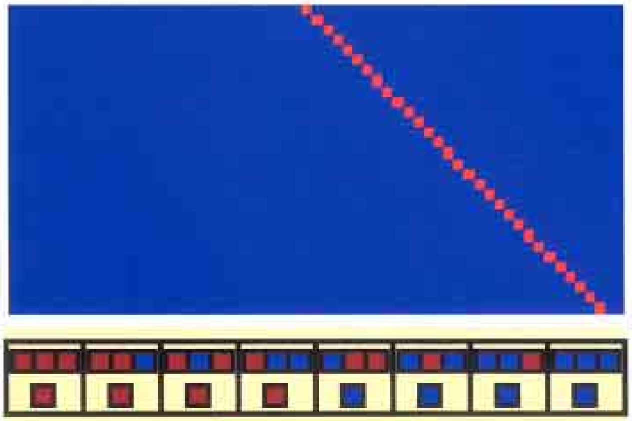
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (u_{i-1})$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i+1| - |x_i-1|)$$



Rule **240**

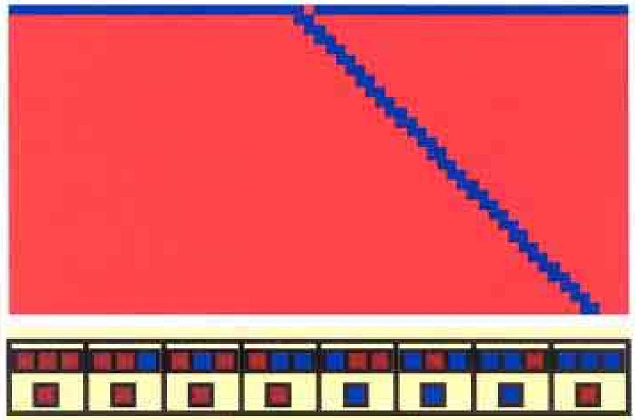
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (2u_{i-1} - u_i - u_{i+1} + 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i+1| - |x_i-1|)$$



Rule **241**

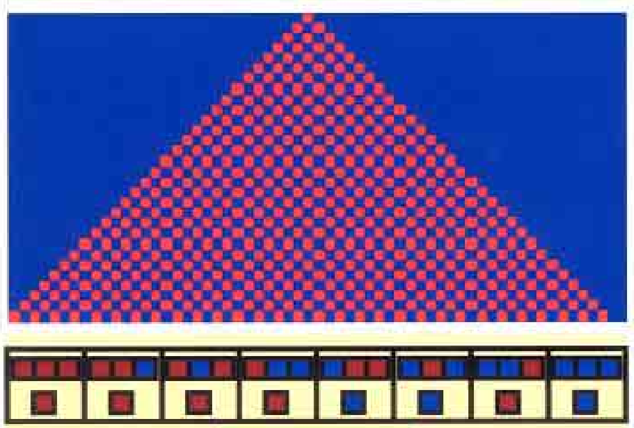
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (2u_{i-1} - u_i + u_{i+1} + 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i+1| - |x_i-1|)$$



Rule **242**

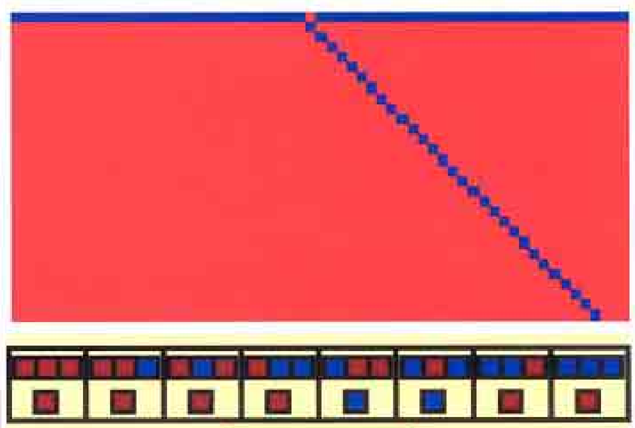
$\kappa = 1$

State Equation

$$\dot{x}_i = -x_i + 2y_i + (u_{i-1} - u_i + 1)$$

Output Equation

$$y_i = y(x_i) \triangleq \frac{1}{2}(|x_i+1| - |x_i-1|)$$



Rule **243**

Table 2. (*Continued*)

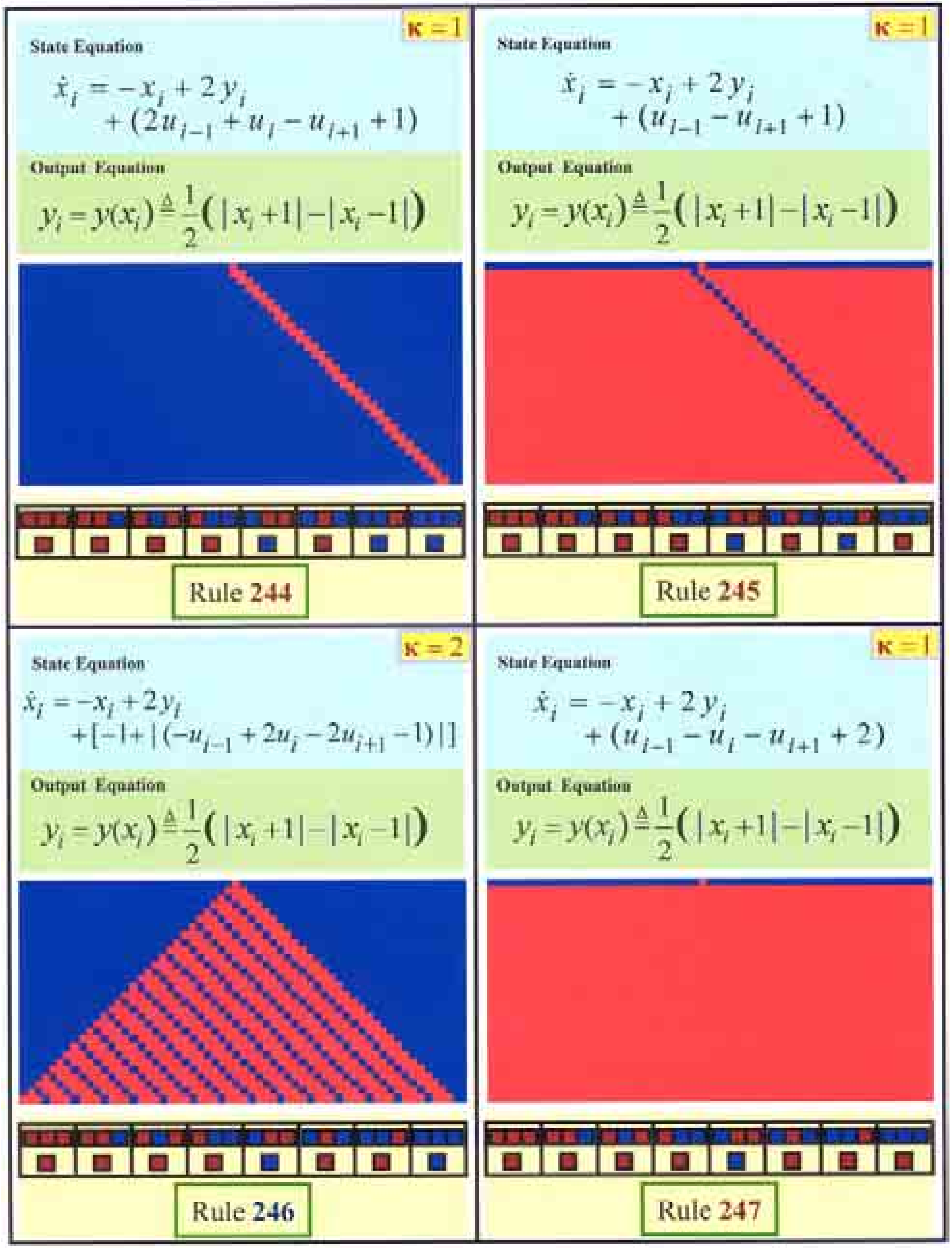


Table 2. (*Continued*)

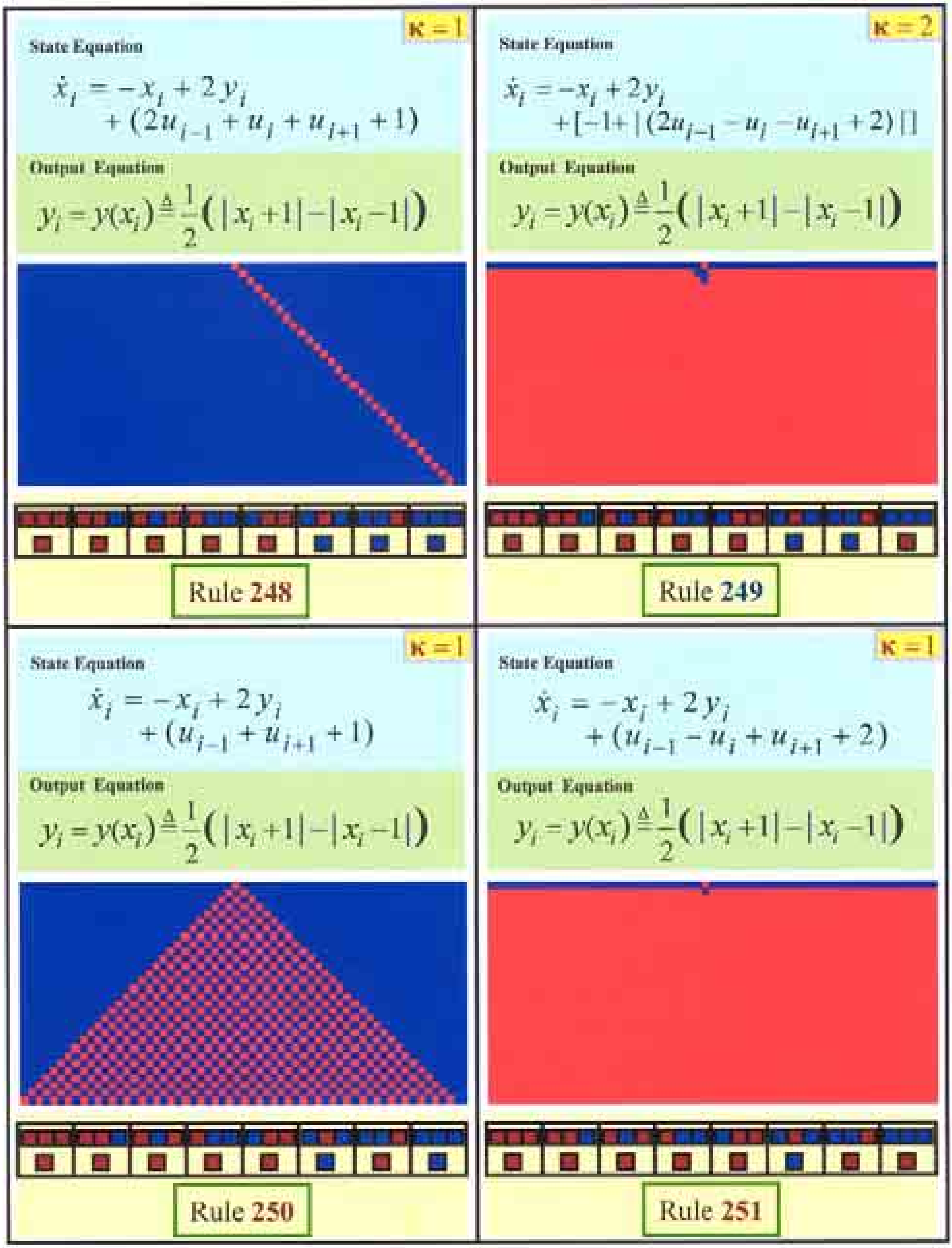
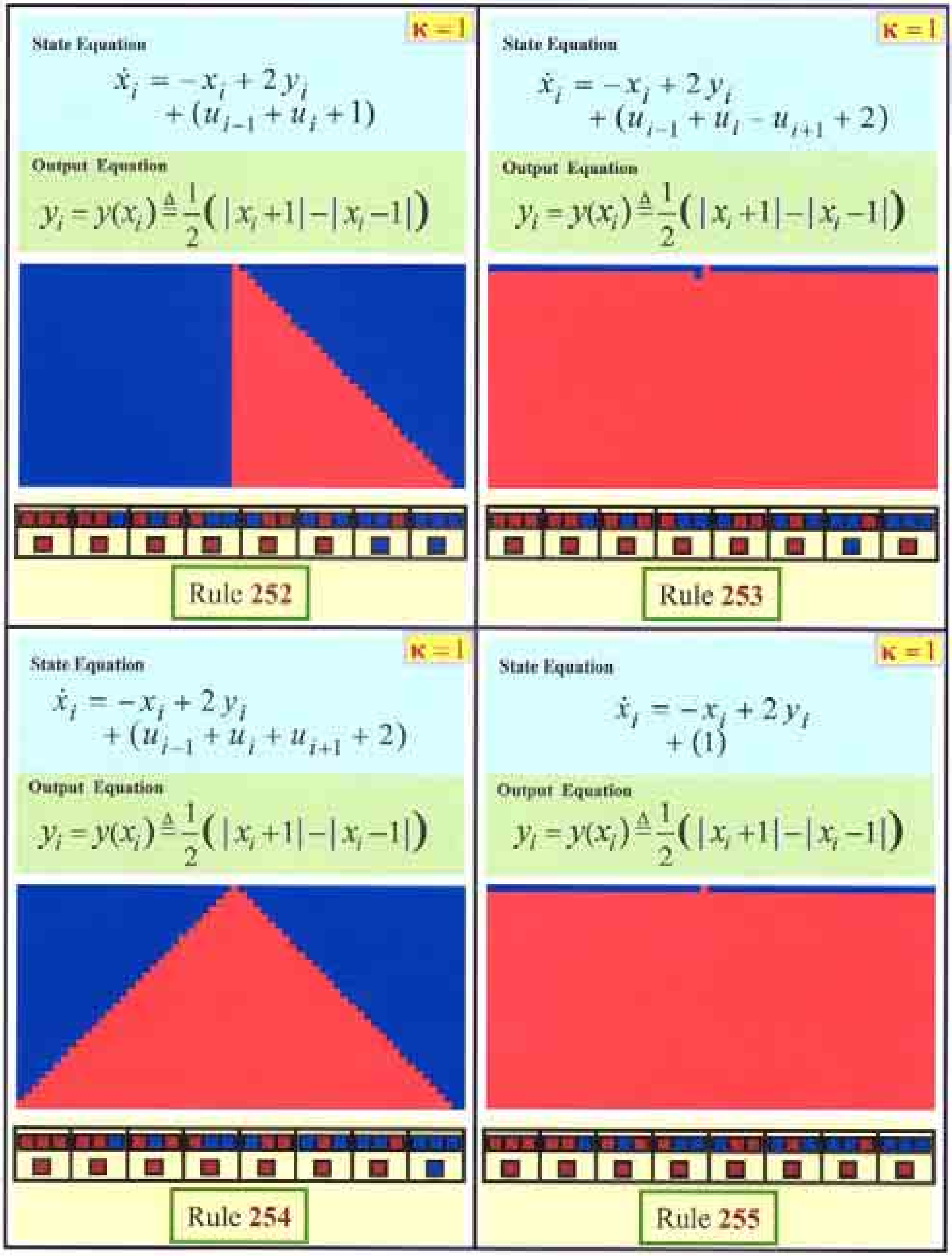


Table 2. (*Continued*)



output variable y_i in the *state equation*. The *initial conditions* for all 256 dynamical systems listed in Table 2 are *the same*, namely, $x_i(0) = 0$.

Observe that all dynamical systems listed in Table 2 have identical *driving-point* functions $g(x_i)$, defined earlier in Eq. (1); namely,

$$g(x_i) = -x_i + |x_i + 1| - |x_i - 1| \tag{8}$$

Equation (8) can be decomposed into the following set of three linear equations:

$$g(x_i) = \begin{cases} x_i\,, & \text{for} \quad |x_i| \le 1 \\ -x_i + 2\,, & \text{fof} \quad x_i > 1 \\ -x_i - 2\,, & \text{for} \quad x_i < -1 \end{cases} \tag{9}$$

The driving-point function $g(x_i)$ is depicted by the green curve Γ in Fig. 3. The remaining part of the state equation for each dynamical system in Table 2 coincides therefore with the *discriminant function* $w(u_{i-1}, u_i, u_{u+1})$ defined in Eq. (1). Since $w(u_{i-1}, u_i, u_{u+1})$ is a *constant real* number[8] for each of the eight vertices defined in Fig. 2, the state equation for each dynamical system in Table 2 can be recast into the following eight simplified equations, one for each vertex ⓝ, $n = 0, 1, 2, \ldots, 7$:

$$\begin{array}{cccl}
\text{Vertex} & \text{Discriminant} & & \text{Simplified Differential Equation} \\
\textcircled{n} & w(u_{i-1}, u_i, u_{i+1}) & & \dot{x}_i = h_n(x_i) \\
0 & w(0) \triangleq w(-1,-1,-1) & \Rightarrow & \dot{x}_i = g(x_i) + w(0) \triangleq h_0(x_i) \\
1 & w(1) \triangleq w(-1,-1,1) & \Rightarrow & \dot{x}_i = g(x_i) + w(1) \triangleq h_1(x_i) \\
2 & w(2) \triangleq w(-1,1,-1) & \Rightarrow & \dot{x}_i = g(x_i) + w(2) \triangleq h_2(x_i) \\
3 & w(3) \triangleq w(-1,1,1) & \Rightarrow & \dot{x}_i = g(x_i) + w(3) \triangleq h_3(x_i) \\
4 & w(4) \triangleq w(1,-1,-1) & \Rightarrow & \dot{x}_i = g(x_i) + w(4) \triangleq h_4(x_i) \\
5 & w(5) \triangleq w(1,-1,1) & \Rightarrow & \dot{x}_i = g(x_i) + w(5) \triangleq h_5(x_i) \\
6 & w(6) \triangleq w(1,1,-1) & \Rightarrow & \dot{x}_i = g(x_i) + w(6) \triangleq h_6(x_i) \\
7 & w(7) \triangleq w(1,1,1) & \Rightarrow & \dot{x}_i = g(x_i) + w(7) \triangleq h_7(x_i)
\end{array} \tag{10}$$

Each of these eight scalar differential equations differs from each other only by a constant. Figure 3 shows two typical cases; the upper curve corresponds to a positive offset of $w(n) > 0$ whereas the lower curve corresponds to a negative offset of $w(n) < 0$. In this context, it is more meaningful to call $w(u_{i-1}, u_i, u_{i+1})$ in Eq. (1) an *offset level.*

Now, since the *initial condition* is, by assumption in Table 2, always equal to $x_i(0) = 0$, the trajectory must begin from the upper initial point $P_+(0)$ if $w(n) > 0$, or from the lower initial point $P_-(0)$ if $w(n) < 0$. Since $\dot{x}_i > 0$ at all points to the right of the initial point $P_+(0)$ on the upper curve, the solution trajectory must "flow" *monotonically* to the right until it arrives at the right equilibrium point Q_+ located at $x_i = \overline{x}_i(Q_+)$. Conversely, the trajectory must begin from the lower initial point $P_-(0)$ if $w(n) < 0$ and flow leftwards until it arrives at the left equilibrium point Q_-. Any *directed* path (indicated by bold arrowheads) on a *translated* driving-point plot is called a *dynamic route* [Chua, 1969]. Once a dynamic route is specified, the steady state value $\overline{x}_i(Q_+)$ at the right equilibrium point, or $\overline{x}_i(Q_-)$ at the left equilibrium point, can be identified by inspection. Observe that $\overline{x}_i(Q_+) > 1$ and $\overline{x}_i(Q_-) < -1$, always!

We are now ready to prove the following fundamental theorem from which Table 2 is generated.

Theorem 2. Explicit Output Formula. *The state $x_i(t)$ of each dynamical system listed in Table 2 with initial condition $x_i(0) = 0$ converges mono-*

[8] Note that even though u_{i-1}, u_i, and u_{i+1} are Boolean variables, they must be treated as *real numbers* here. This is the reason why it is essential to use "−1" instead of "0" in the truth table in Fig. 2.

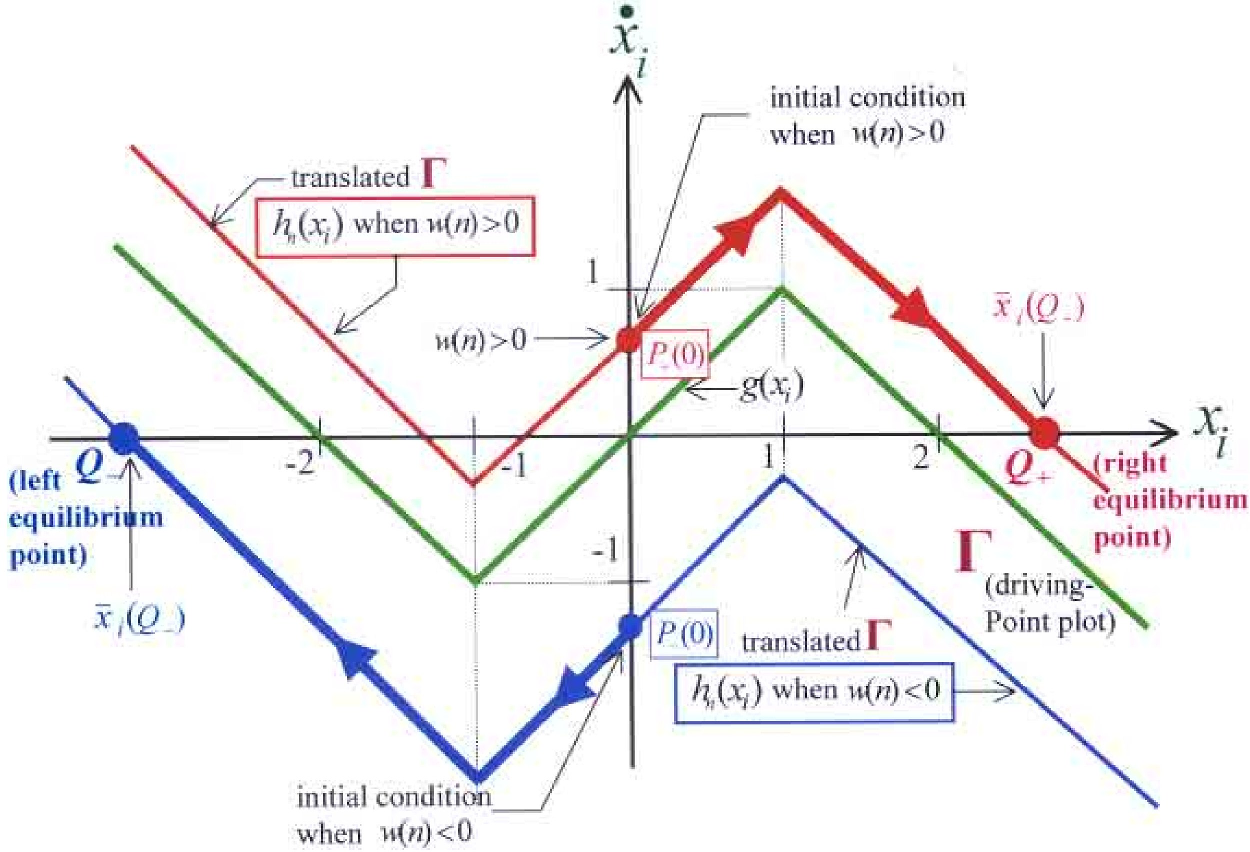


Fig. 3. Green curve Γ denotes the plot of the driving-point function $g(x_i)$ of Eq. (1). Red curve denotes a vertical translation of Γ upward by an offset level equal to $w(n) > 0$, $n = 0, 1, 2, \ldots, 7$. Blue curve denotes a vertical translation of Γ downward by an offset level equal to $w(n) < 0$. Each path with arrowhead is called a *dynamic route* depicting motion from initial point $P_+(0)$ to *attractor* (equilibrium point) Q_+, or from initial point $P_-(0)$ to *attractor* Q_-.

tonically to an attractor Q_+ located at $\overline{x}_i(Q_+) > 1$ for each input (u_{i-1}, u_i, u_{i+1}) which gives rise to a positive offset level $w(u_{i-1}, u_i, u_{i+1}) > 0$, or to an attractor Q_- located at $\overline{x}_i(Q_-) < -1$ for each input which gives rise to a negative offset level $w(u_{i-1}, u_i, u_{i+1}) < 0$.

The corresponding *output* $y_i(t)$ converges to the Boolean state $y_i = 1$ in the former, and to the Boolean state $y_i = -1$ in the latter case. Moreover, the steady-state *output* of Eq. (1) at equilibrium is given explicitly by the formula (for initial condition $x_i(0) = 0$)

$$y_i = \text{sgn}\{w(\sigma)\} \tag{11}$$

for *any*[9] discriminant function $w(\sigma) \triangleq w(u_{i-1}, u_i, u_{i+1})$. For the particular $w(\sigma)$ given in Table 2, the output *at equilibrium* is given explicitly by:

Attractor Color Code

$$y_i = \text{sgn}\{z_2 \pm |[z_1 \pm |z_0 + (b_1 u_{i-1} + b_2 u_i + b_3 u_{i+1}|]|\} \tag{12}$$

Proof. Since the driving-point plot Γ in Fig. 3 can shift only up or down by an amount equal to the offset level $w(u_{i-1}, u_i, u_{i+1})$, it follows from the geometrical construction in Fig. 3 that $\overline{x}_i(Q_+) > 1$ and $\overline{x}_i(Q_-) < -1$. Moreover, since *the initial condition* is located at $P_+(0)$ if $w(n) > 0$, or at $P_-(0)$ if $w(n) < 0$, it follows from the *dynamic route* spec-

[9]Since the proof of Theorem 2 is independent of the choice of the discriminant function $w(u_{i-1}, u_i, u_{i+1})$, it follows that forumula (11) is valid not only for the dynamical systems listed in Table 2 which are defined in terms of *absolute-value functions*, but for *any* other discriminant function.

ified (with arrowheads) in Fig. 3 that $x_i(t)$ must increase monotonically to Q_+ if $w(n) > 0$, and must decrease monotonically to Q_- if $w(n) < 0$. Finally, it follows from Eq. (7) that $y_i \to +1$ whenever $\overline{x}_i(Q_+) > 0$, or $y_i \to -1$ whenever $\overline{x}_i(Q_-) < 0$.

An exhaustive analysis of the dynamic routes in Fig. 3 for each of the eight vertices shows that at equilibrium,

$$y_i \to +1 \quad \text{if} \quad w(\sigma) > 0$$

$$y_i \to -1 \quad \text{if} \quad w(\sigma) < 0$$

where $\sigma \triangleq b_1 u_{i-1} + b_2 u_i + b_3 u_{i+1} = \mathbf{b}^T \mathbf{u}$. It follows that

$$y_i \to \text{sgn}\{w(u_{i-1}, u_i, u_{i+1})\} \quad \blacksquare \qquad (13)$$

Table 2 consists of 64 pages, each page contains the *dynamical system* and the *local rule* it encodes, each one identified by its Rule number N, $N = 0, 1, 2, \ldots, 255$. The truth table for each rule N is generated by the associated *dynamical system* defined in the upper portion of each quadrant, and not from the truth table, thereby proving that each dynamical system and the local rule it encodes are one and the same. The truth table for each rule in Table 2 is cast in the format of a "gene decoding book"[10] with only $2^{2^3} = 256$ distinct 1×3 "*neighborhood patterns*". These patterns are ordered from right to left starting with (0 0 0) as the least significant binary bit, as in Fig. 39 of [Chua, 1998]. With this convention, the string of eight 1×3 patterns (also called a "*gene decoding tape*" in [Chua, 1998]) shown directly above each Rule N in Table 2 is redundant (they are the same for all Rules) but is included for ease of reference, as well as for comparison with examples in [Wolfram, 2002], where the same format is used. Each color picture in Table 2 consists of 30×61 pixels, generated by a one-dimensional Cellular Automata (with 61 cells and a periodic boundary condition) *with a* specified local *Rule N*. For ease of comparison, we have adopted the format used in pages 55–56 of [Wolfram, 2002] where the top row corresponds to the *initial* pattern, which is "0" (blue in Table 2) in all pixels except the *center* pixel (labeled as cell 0 in Fig. 1) which is "1" (red in Table 2). The evolution over the next 29 iterations is conveniently displayed in rows 2 to 30, as in [Wolfram, 2002]. A comparison of each pattern in Table 2 (which is generated from a corresponding dynamical system) with the corresponding pattern in [Wolfram, 2002] (which is generated from a truth table) shows that they are identical. This identification procedure provides therefore a conceptually simple and constructive yet completely rigorous proof that each dynamical system in Table 2 and the local *rule* it encodes are one and the same.

4.1. *Dynamical System for Rule 110*

For concreteness, let us examine one of the local rules from Table 2, namely, Wolfram's celebrated Rule 110, the simplest *universal Turing machine* known to date. The differential equation extracted from Rule 110 in Table 2 is:

Differential Equation for Rule 110

$$\dot{x}_i = (-x_i + |x_i + 1| - |x_i - 1|) + [-2 + |(u_{i-1} + 2u_i - 3u_{i+1} - 1)|]$$
$$x_i(0) = 0 \qquad (14)$$

Using the notation introduced in Eqs. (3) and (5), we can identify the following relevant data for Rule 110:

Data for Rule 110

Projection:

$$\sigma = u_{i-1} + 2u_i - 3u_{i+1}$$

Orientation vector:

$$\mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix}$$

Discriminant:

$$w(\sigma) = -2 + |\sigma - 1| \qquad (15)$$

Note that $w(\sigma)$ corresponds to Eq. (5) with $z_2 = -2$, $z_1 = 0$, $z_0 = -1$ and with the positive sign adopted at both locations. The equilibrium solution of the Differential Equation (14) gives the *attractor*

[10] The *gene decoding book* for the well-known two-dimensional Cellular Automata called *Game of Life* is cast in this format on pages 145–152 of [Chua, 1998]. It has $2^9 = 512$ distinct 3×3 *neighborhood patterns*.

color code via the output equation (12) for each vertex ⓝ of the Boolean cube no. 110 in Table 1; namely,

Attractor Color Code for Rule 110

$$y_i = \mathrm{sgn}[-2 + |(u_{i-1} + 2u_i - 3u_{i+1} - 1)|] \quad (16)$$

Substituting the input (u_{i-1}, u_i, u_{i+1}) of each vertex ⓝ, $n = 0, 1, 2, \ldots, 7$, to Eq. (16), we obtain the following eight simplified Differential Equations for Rule 110:

Vertex	Discriminant		Simplified Differential Equation	Attractor $\overline{x}_i(Q)$	Color of Vertex
ⓝ	$w(u_{i-1}, u_i, u_{i+1})$		$\dot{x}_i = h_n(x_i)$		ⓝ
0	$w(-1,-1,-1) = -2 + \|(-1-2+3-1)\| = -1$	$\Rightarrow$	$\dot{x}_i = g(x_i) - 1$	-3	■
1	$w(-1,-1,1) = -2 + \|(-1-2-3-1)\| = 5$	$\Rightarrow$	$\dot{x}_i = g(x_i) + 5$	7	■
2	$w(-1,1,-1) = -2 + \|(-1+2+3-1)\| = 1$	$\Rightarrow$	$\dot{x}_i = g(x_i) + 1$	3	■
3	$w(-1,1,1) = -2 + \|(-1+2-3-1)\| = 1$	$\Rightarrow$	$\dot{x}_i = g(x_i) + 1$	3	■
4	$w(1,-1,-1) = -2 + \|(1-2+3-1)\| = -1$	$\Rightarrow$	$\dot{x}_i = g(x_i) - 1$	-3	■
5	$w(1,-1,1) = -2 + \|(1-2-3-1)\| = 3$	$\Rightarrow$	$\dot{x}_i = g(x_i) + 3$	5	■
6	$w(1,1,-1) = -2 + \|(1+2+3-1)\| = 3$	$\Rightarrow$	$\dot{x}_i = g(x_i) + 3$	5	■
7	$w(1,1,1) = -2 + \|(1+2-3-1)\| = -1$	$\Rightarrow$	$\dot{x}_i = g(x_i) - 1$	-3	■

(17)

As always, the *driving-point function* $g(x_i)$ in Eq. (17) is *invariant* for all Rules in Table 2, and is given by Eq. (2).

Note that the color of the eight vertices in the Boolean cube no. 110 in Table 1 is identical to that predicted in Eq. (17), as expected.

4.2. *There are Eight Attractors for Each Local Rule*

Our preceding in-depth analysis of the nonlinear dynamics of Eqs. (1) and (2) via Eq. (10) and Fig. 3 shows that there may be two attractors (e.g. upper curve in Fig. 3 has 2 attractors and 1 repellor) in the one-dimensional, x_i-state space for each input (u_{i-1}, u_i, u_{i+1}); namely, one located at $\overline{x}_i(Q_+) > 0$ and the other located at $\overline{x}_i(Q_-) < 0$. Since there are eight inputs corresponding to the eight vertices of each Boolean cube in Table 1, it appears that there may be 16 attractors for some local rules. This observation is counter-intuitive because one would expect that since there are eight vertices, one would need only eight attractors for each local rule.

To resolve the above paradox, abserve that our imposition of the initial condition $x_i(0) = 0$

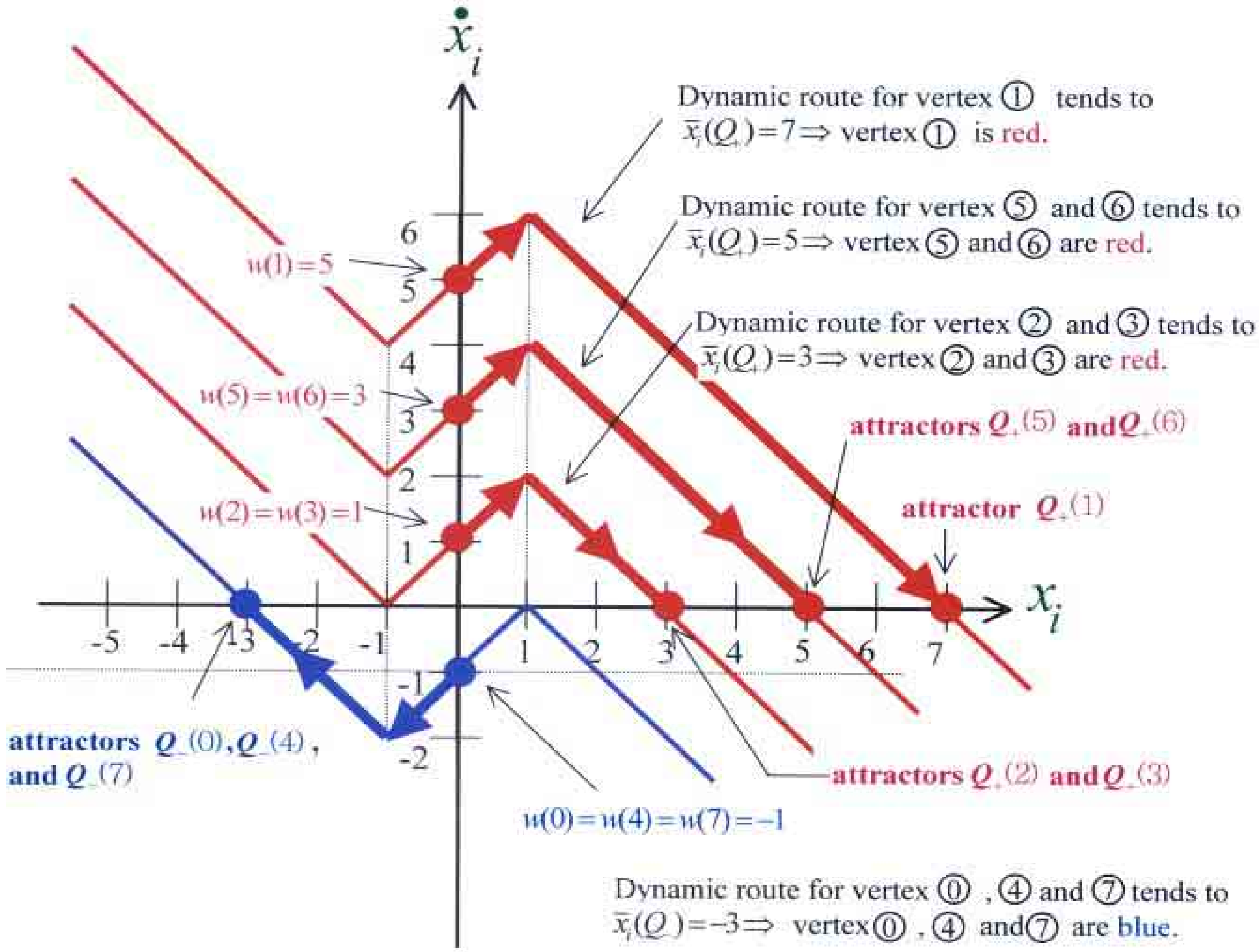


Fig. 4. Dynamic routes for local Rule 110. The five initial points $w(1)$, $w(2)$, $w(3)$, $w(5)$, and $w(6)$ lying above the x_i-axis converge to $Q_+(1)$, $Q_+(2)$, $Q_+(3)$, $Q_+(5)$, and $Q_+(6)$ respectively, implying these five vertices must be coded in red. The remaining three initial $w(0)$, $w(4)$, and $w(7)$ lying below the x_i-axis converge to attractors $Q_-(0)$, $Q_-(4)$, and $Q_-(7)$, respectively, implying these three vertices must be coded in blue.

ensures that only *one* attractor is relevant for *each* input.

It is sometimes convenient to interpret the three input variables u_{i-1}, u_i, and u_{i+1} also as *state variables* by defining the following *equivalent* dynamical system in $\mathbf{R}^4$:

$$\begin{aligned} \dot{x}_i &= g(x_i) + w(u_{i-1}, u_i, u_{i+1}) \\ \dot{u}_{i-1} &= 0 \\ \dot{u} &= 0 \\ \dot{u}_{i+1} &= 0 \end{aligned} \tag{18}$$

Observe that Eqs. (1) and (18) have identical solutions if we choose the following initial conditions:

$$\begin{aligned} x_i(0) &= 0 \\ u_{i-1}(0) &\in \{-1, 1\} \\ u_i(0) &\in \{-1, 1\} \\ u_{i+1}(0) &\in \{-1, 1\} \end{aligned} \tag{19}$$

The solutions of Eq. (18) consist of a continuum of straight lines parallel to the x_i-axis in the four-dimensional Euclidean space $\mathbf{R}^4$, as depicted in Fig. 5(a) for the special case of only two inputs u_i and u_{i+1}, thereby allowing a geometrical visualization in $\mathbf{R}^3$ with the x_i-axis pointing outward from the paper. We can interpret this geometrical structure as a special case of the truth table in Fig. 2 where u_{i-1} is fixed at $u_{i-1} = 1$ thereby reducing it to only the last four rows corresponding to vertices ④, ⑤, ⑥ and ⑦, equivalent to the truth table of the XOR operation between the two binary variables u_i and u_{i+1}. In this reduced setting, all trajectories of the dynamical system (18) are *parallel straight lines* passing through every point (u_i, u_{i+1}) within any

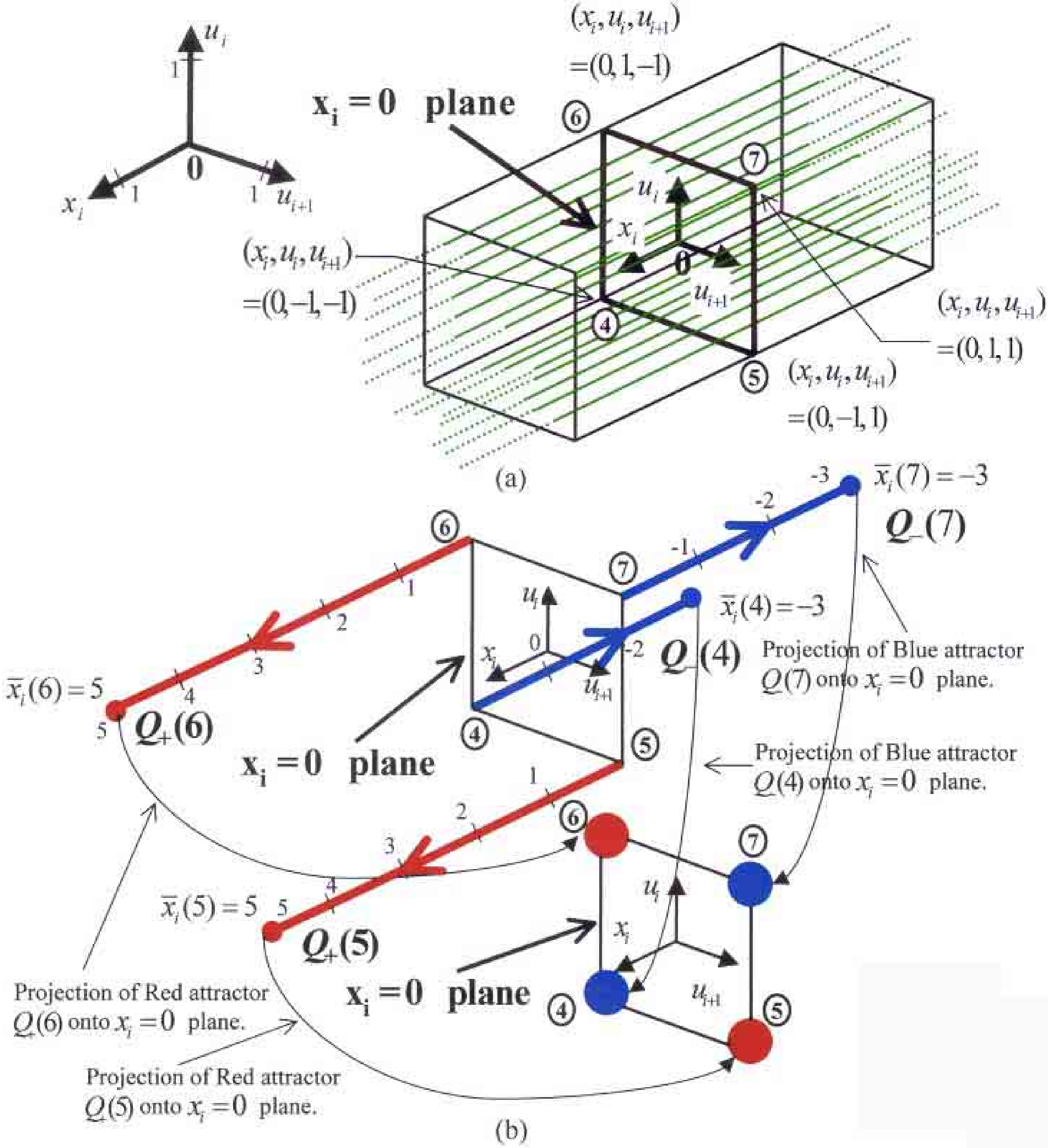


Fig. 5. (a) A three-dimensional state space with coordinates (x_i, u_i, u_{i+1}). A square cross-section at $x_i = 0$ is highlighted with vertices corresponding to the front face of the Boolean cube in Table 1. Each parallel line is a *trajectory* with constant u_i and u_{i+1}. (b) If we assume in $u_{i-1} = 1$ in Rule 110, we can visualize four attractors $Q_+(6)$, $Q_+(5)$, $Q_-(4)$, and $Q_-(7)$ whose x_i-coordinate coincides with that from Fig. 4.

square cross-section of a Boolean cube in Table 1, as depicted in Fig. 5(a) at $x_i = 0$. In other words, the general solutions of Eq. (18) look like a bundle of parallel fibers within an assumed square cross-section at $x_i = 0$.

However, since we are only interested in *binary* cellular automata in this paper, we need to examine only the fibers through the four vertices ④, ⑤, ⑥ and ⑦. An inspection of the dynamic routes through these four vertices in Fig. 4 leads to the four attractors $Q_-(4)$, $Q_+(5)$, $Q_+(6)$, and $Q_-(7)$; they are located along each respective fiber with an x_i-coordinate value equal to $\overline{x}_i(4) = -3$, $\overline{x}_i(5) = 5$, $\overline{x}_i(6) = 5$, and $\overline{x}_i(7) = -3$, respectively, as shown in Fig. 5(b). By our earlier color code, the two attractors $Q_+(5)$ and $Q_+(6)$ would be coded in *red*, whereas the other 2 attractors $Q_-(4)$ and $Q_-(7)$ would be coded in *blue*. Observe that we now have exactly $2^2 = 4$ attractors when there are only two inputs, and there will be $2^3 = 8$ attractors when there are three binary inputs u_{i-1}, u_i, and u_{i+1}. In general, we will have 2^n attractors for n binary inputs. For example, in a two-dimensional cellular automata with eight nearest neighbors, such as the *Game of Life*, we would have $2^9 = 512$ attractors corresponding to the 512 vertices of a Boolean cube in a nine-dimensional Euclidean space.

In view of the above one-to-one correspondence between the *number of attractors* of a *dynamical system* representation of a local Rule, and the *number of vertices* of its associated Boolean cube, there is no loss of generality for us to use the *color* at each vertex of a Boolean cube to encode the fourth coordinate x_i of each attractor, as we have done in the bottom portion of Fig. 5(b). In other words, each Boolean cube in Table 1 now encodes the four

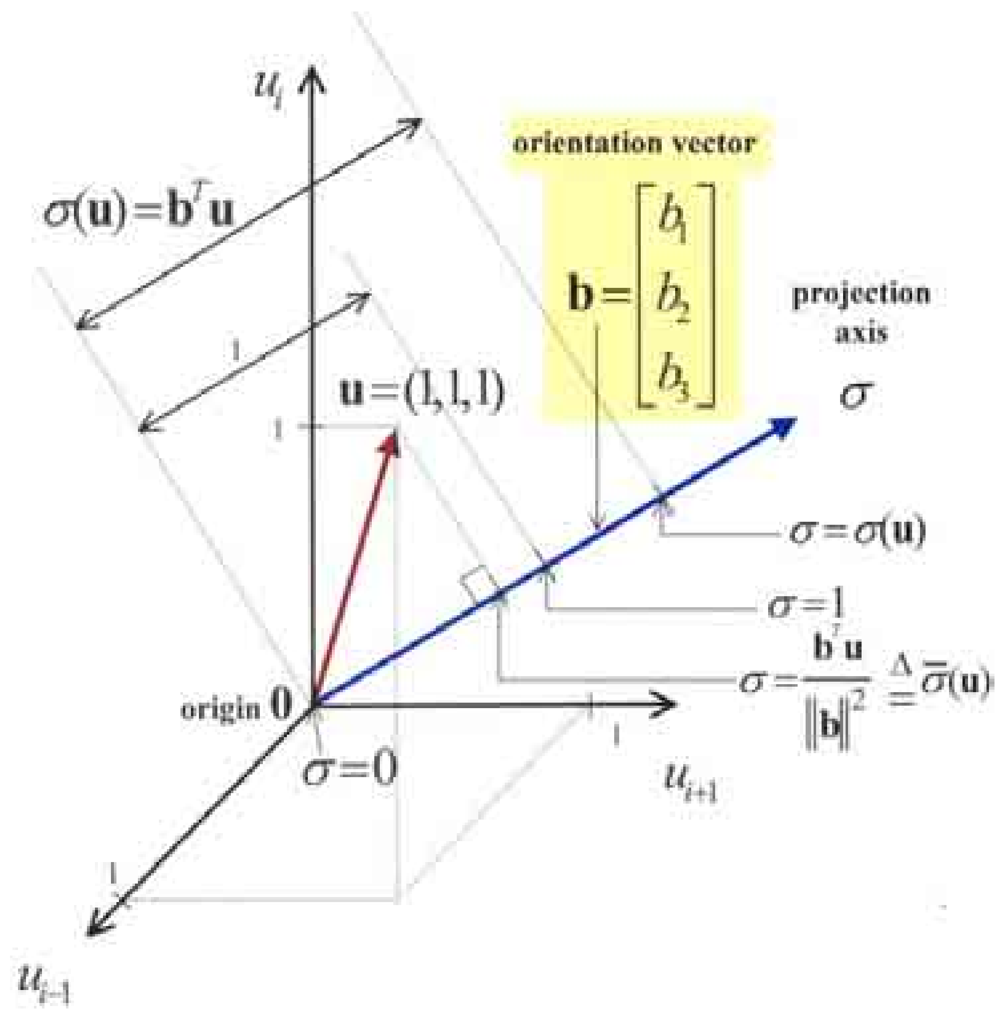


Fig. 6. Geometrical interpretation of the scalar projection $\sigma(\mathbf{u}) \triangleq \mathbf{b}^T\mathbf{u}$ along the projection axis σ. Note that the length $\overline{\sigma}(\mathbf{u})$ of the perpendicular projection of each vertex $\mathbf{u} = (u_{i-1}, u_i, u_{i+1})$ of the Boolean cube onto the projection axis (which coincides with the orientation vector b) is equal to dividing $\sigma(\mathbf{u})$ by $\|\mathbf{b}\|^2 = (b_1^2 + b_2^2 + b_3^2)$.

Table 3. Calculation of $y_i = \text{sgn}[w(\sigma)]$ for Rule 110 at vertex ⓝ with $\mathbf{b} = [1 \quad 2 \quad -3]^T$.

n	u_{i-1}	u_i	u_{i+1}	$\bar{\sigma}(n)$	$\sigma(n)$	$w(n)$	y_i
0	-1	-1	-1	0	0	-1	-1
1	-1	-1	1	-0.43	-6	5	1
2	-1	1	-1	0.29	4	1	1
3	-1	1	1	-0.15	-2	1	1
4	1	-1	-1	0.15	2	-1	-1
5	1	-1	1	-0.29	-4	3	1
6	1	1	-1	0.43	6	3	1
7	1	1	1	0	0	-1	-1

binary coordinates $(x_i, u_{i-1}, u_i, u_{i+1})$, where x_i is assumed to be a *binary* variable after thresholding via the output equation (7).

5. Every Local Rule has a Unique Complexity Index

Theorem 2 from the preceding section, as well as Table 2, provided a constructive proof that every local rule can be generated by a *dynamical system* whose attractors have a one-to-one *correspondence* with the color of the vertices of a Boolean cube which encodes the corresponding truth table in Table 1. Perhaps the most surprising result from Theorem 2 is the implication of Eq. (11), which asserts that each local Rule N depends only on the *single* scalar projection variable σ defined in Eq. (3), regardless of the number of inputs.[11] Moreover, since the sgn(•) function is determined by the sign of $w(\sigma)$, it follows that the binary output y_i, or equivalently, the color (red or blue) of each vertex ⓝ of *each* local rule depends entirely on the *discriminant* function $w(\sigma)$. The fact that this result should hold for *all* local rules is hard to believe but it is true! It follows that the key to characterize the properties and complexity of a local Rule is to analyze the *structure* of the discriminant $w(\sigma)$ as a function of σ.

5.1. *Geometrical Interpretation of Projection σ and Discriminant $w(\sigma)$*

Since $\mathbf{u} = (u_{i-1}, u_i, u_{i+1})^T$ is simply a vector from the origin (i.e. center of the Boolean cube) to one

[11] Since the proof of Theorem 2 never invokes in the individual inputs, but only on $\sigma = \mathbf{b}^T\mathbf{u}$, it follows that Theorem 2 holds for any number of inputs, including $n = 9$ in two-dimensional Cellular Automata with eight nearest neighbors.

of the eight vertices of the cube, it follows that $\sigma = \mathbf{b}^T\mathbf{u} = b_1u_{i-1} + b_2u_i + b_3u_{i+1}$ is just the *projection* of $\mathbf{u}$ onto the *orientation* vector $\mathbf{b}$, as depicted in Fig. 6. To simplify our following geometrical interpretation, it is convenient to examine the *normalized* projection $\overline{\sigma}(\mathbf{u})$ obtained by dividing $\sigma(\mathbf{u})$ by $\|\mathbf{b}\|^2 = b_1^2 + b_2^2 + b_3^2$. Since $\|\mathbf{u}\| = \sqrt{3} > 1$, for any vertex ⓝ, it follows that the normalized projection $\overline{\sigma}$ stays inside the cube. In particular, we can calculate the value of $\overline{\sigma}(\mathbf{u})$ with respect to each *vertex* ⓝ, $n = 0, 1, 2, \ldots, 7$ and mark the value of this *normalized* projection along the normalized orientation axis $\overline{\sigma}$ defined by $[b_1/\|\mathbf{b}\|^2 \quad b_2/\|\mathbf{b}\|^2 \quad b_3/\|\mathbf{b}\|^2]^T$, as shown in Fig. 7(a) for Rule 110 which we have just analyzed in the preceding section. The calculated values of $\overline{\sigma}(\mathbf{u})$ for Rule 110 are given in Table 3, along with the value of the *discriminant* $w(\sigma)$ corresponding to each vertex ⓝ, $n = 0, 1, 2, \ldots, 7$. Each value of $\overline{\sigma}(\mathbf{u})$ is identified by a *cross* on the normalized projection axis $\overline{\sigma}$, whose distance from the origin is equal to the calculated value in Table 3. Observe that the position and direction of $\overline{\sigma}$ is determined uniquely by the vector $\mathbf{b} = [b_1 \quad b_2 \quad b_3]^T$, which has been named the *orientation vector* to emphasize its important role in the partitioning of the normalized projection axis $\overline{\sigma}$. Observe also that each cross on $\overline{\sigma}$ in Fig. 7(a) inherits the color of its associated vertex.

Let us redraw the partitioned axis in the usual horizontal position, as shown in Fig. 7(b), and revert back to the original "unscaled" projection axis σ.[12]

The final and crucial step of our analysis consists of plotting the discriminant function $w(\sigma) = -2 + |\sigma - 1|$ for *Rule 110*, as shown in Fig. 7(b). Observe that the resulting discriminant curve $w(\sigma)$ has *separated* the *red* projection crosses from the *blue* projection crosses in such a way that $w(\sigma) > 0$ for each σ associated with a *red cross* (shown directly below vertex no. ①, ⑤, ③, ② and ⑥ in Fig. 7(b)) and $w(\sigma) < 0$ at the remaining crosses, namely the three blue crosses identified with vertex ⓪, ④, and ⑦. The calculated values of the discriminant for Rule 110 are given in Table 4.

An examination of Fig. 7(b) reveals the presence of a *zero-crossing*, or *transition point*, whenever a cross from a nearest-neighbor changes sign. It follows that the selection of an appropriate *discriminant* function having a desired set of transition points provides us with a simple algorithm for synthesizing *any* local Rule from Table 2.

5.2. *Geometrical Interpretation of Transition Points of Discriminant Function $w(\sigma)$*

Since $\sigma \triangleq b_1u_{i-1} + b_2u_i + b_3u_{i+1}$, each *transition point* $\sigma = \sigma_k$ that satisfies $w(\sigma_k) = 0$ ($\sigma_1 = -1$ and $\sigma_2 = 3$ in Fig. 7(b)) defines a two-dimensional plane

$$b_1u_{i-1} + b_2u_i + b_3u_{i+1} = \sigma_k \tag{20}$$

in the three-dimensional (u_{i-1}, u_i, u_{i+1}) *input space*, henceforth called a *separation plane*. For example, the two separation planes associated with the two transition points $\sigma_1 = -1$ and $\sigma_2 = 3$ for Rule 110 with respect to the discriminant function $w(\sigma) = -2 + |\sigma - 1|$ shown in Fig. 7(b) are as follow:

Equation of Separation plane through Transition point $\sigma = \sigma_1 = -1$:	Equation of Separation plane through Transition point $\sigma = \sigma_2 = 3$:
$u_{i-1} + 2u_i - 3u_{i+1} = -1$	$u_{i-1} + 2u_i - 3u_{i+1} = 3$

(21)

These two separation planes are sketched in Fig. 8 and identified by light blue and yellow colors, respectively. We have superimposed on top of these two pictures also the *unscaled orientation vector* $\sigma(\mathbf{u}) = u_{i-1} + 2u_i - 3u_{i+1}$ to show that the yellow separation plane is 3 units above the origin along the *positive* direction of the projection axis σ (*northwest* direction in this case) and that the light blue separation plane is 1 unit below the origin. It is clear from Fig. 8 and Eq. (21) that the

[12]Note that the *unscaled* projection σ should always be used in any computation or design involving the discriminant $w(\sigma)$. The normalized projection $\overline{\sigma}$ is used in Fig. 7(a) to allow visualization on approximately the same scale as $\mathbf{u}$. To avoid cumbersome notations, we will sometimes use the *scalar* $\sigma(\mathbf{u})$ to denote the vector $\sigma(\mathbf{u})[\mathbf{b}/(\mathbf{b}\bullet\mathbf{b})]$, as in Fig. 8.

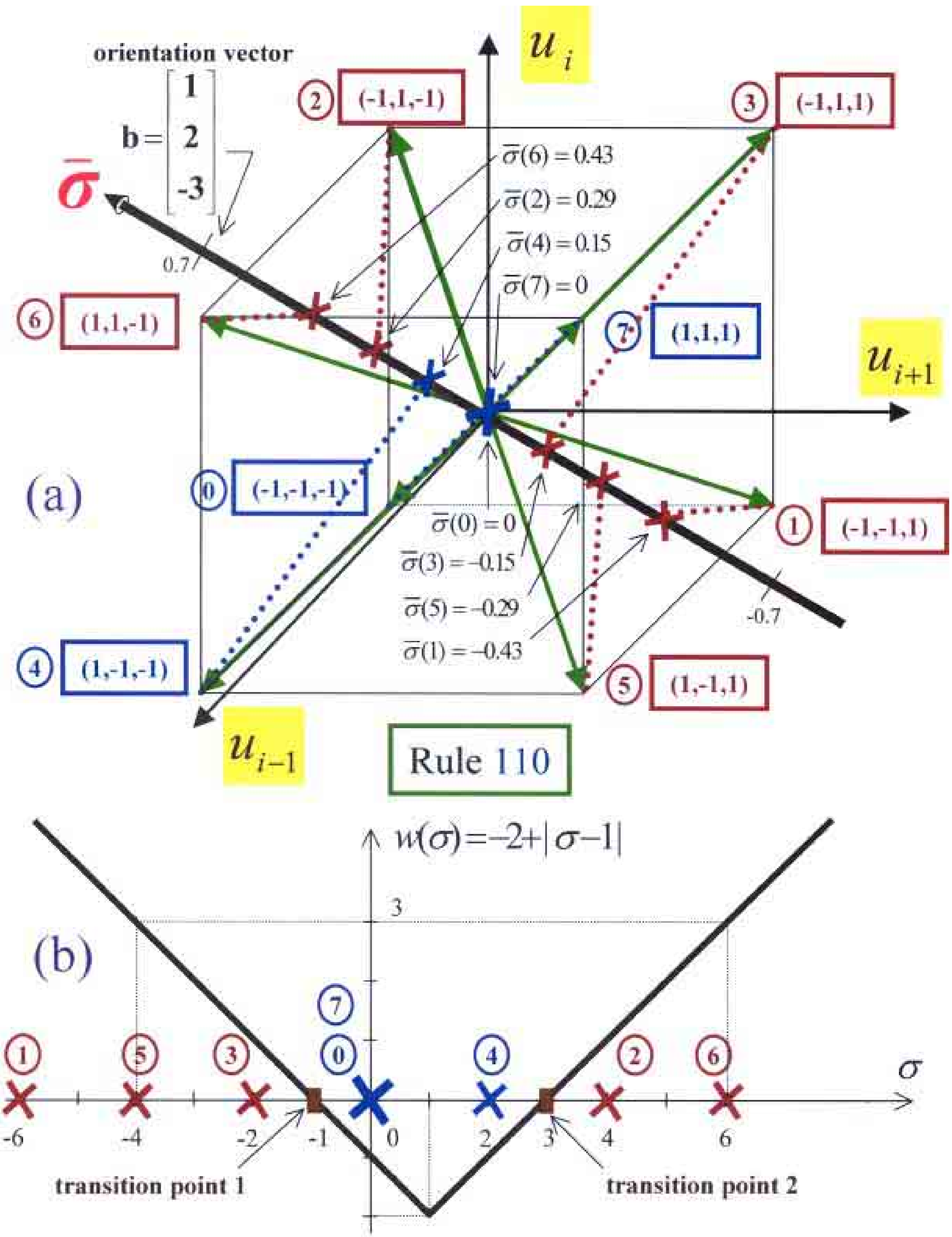


Fig. 7. (a) Projection of each vertex onto the normalized $\overline{\sigma}$-axis defined by $\mathbf{b} = [1 \quad 2 \quad -3]^T$. Each projection $\overline{\sigma}(n)$ obtained from Table 3 is marked by a cross bearing the same color as its associated vertex ⓝ. Note that vertices ⓪ and ⑦ both project into the origin. This "double" cross is depicted in (b) by a *larger* cross. (b) Actual projection $\sigma = \mathbf{b}^T\mathbf{u}$ of each vertex redrawn with separating curve $w(\sigma)$ shows two transition points, thereby requiring two parallel separating planes.

Table 4. Calculation of the discriminant $w(\sigma) = -2 + |(u_{u-1} + 2u_i - 3u_{i+1}) - 1|$ at vertex ⓝ.

n	$w(\sigma) = -2 + \|(u_{i-1} + 2u_i - 3u_{i+1}) - 1\| \triangleq -2 + \|\sigma - 1\|$
0	$w(0) = -2 + \|(-1-2+3)-1\| = -2 + \|-1\| = -1$
1	$w(1) = -2 + \|(-1-2-3)-1\| = -2 + \|-7\| = 5$
2	$w(2) = -2 + \|(-1+2+3)-1\| = -2 + \|3\| = 1$
3	$w(3) = -2 + \|(-1+2-3)-1\| = -2 + \|-3\| = 1$
4	$w(4) = -2 + \|(1-2+3)-1\| = -2 + \|1\| = -1$
5	$w(5) = -2 + \|(-1-2-3)-1\| = -2 + \|-5\| = 3$
6	$w(6) = -2 + \|(1+2+3)-1\| = -2 + \|5\| = 3$
7	$w(7) = -2 + \|(1+2-3)-1\| = -2 + \|-1\| = -1$

two separation planes are not only parallel to each other, but they are also perpendicular to the orientation vector $\mathbf{b} = [1 \quad 2 \quad -3]^T$. The orthogonality relationship between a separation plane defined by $\mathbf{b}^T\mathbf{u} = \sigma_k$ and its associated *orientation vector* $\mathbf{b}$ is also obvious if one views σ as one coordinate axis of a rotated coordinate system through the origin so that $\sigma = \sigma_k$ is, by definition of a coordinate axis, the plane perpendicular to the σ-axis at the point $\sigma = \sigma_k$.

A careful inspection of Fig. 8 reveals the *raison etre* for introducing the abstract concept of *separation planes*;[13] namely, *they separate the vertices into clusters of the same color.*

5.3. *Geometrical Structure of Local Rules*

The colored vertex separating ability of the separation planes is a truly *fundamental geometric property* of the Euclidean Space, independent of the choice of the basis function used to describe the *discriminant* function $w(\sigma)$, so long as $w(\sigma)$ has the same transition points. For example, we could have chosen

$$w(\sigma) = (\sigma - 3)(\sigma - 1) = \sigma^2 - 4\sigma + 3 \qquad (22)$$

and substituted it into Eq. (1) to obtain another dynamical system

Alternate State Equation for Rule 110

$$\dot{x}_i = g(x_i) + (u_{i-1} + 2u_i - 3u_{i+1})^2 - 4(u_{i-1} + 2u_i - 3u_{i+1}) + 3 \qquad (23)$$

that would generate exactly *the same local Rule*; namely, Rule 110 in this case. Indeed, instead of using absolute-value functions in Table 2, we could

[13] The concepts of *transition points* and *separation planes* were first introduced in [Dogaru & Chua, 1999].

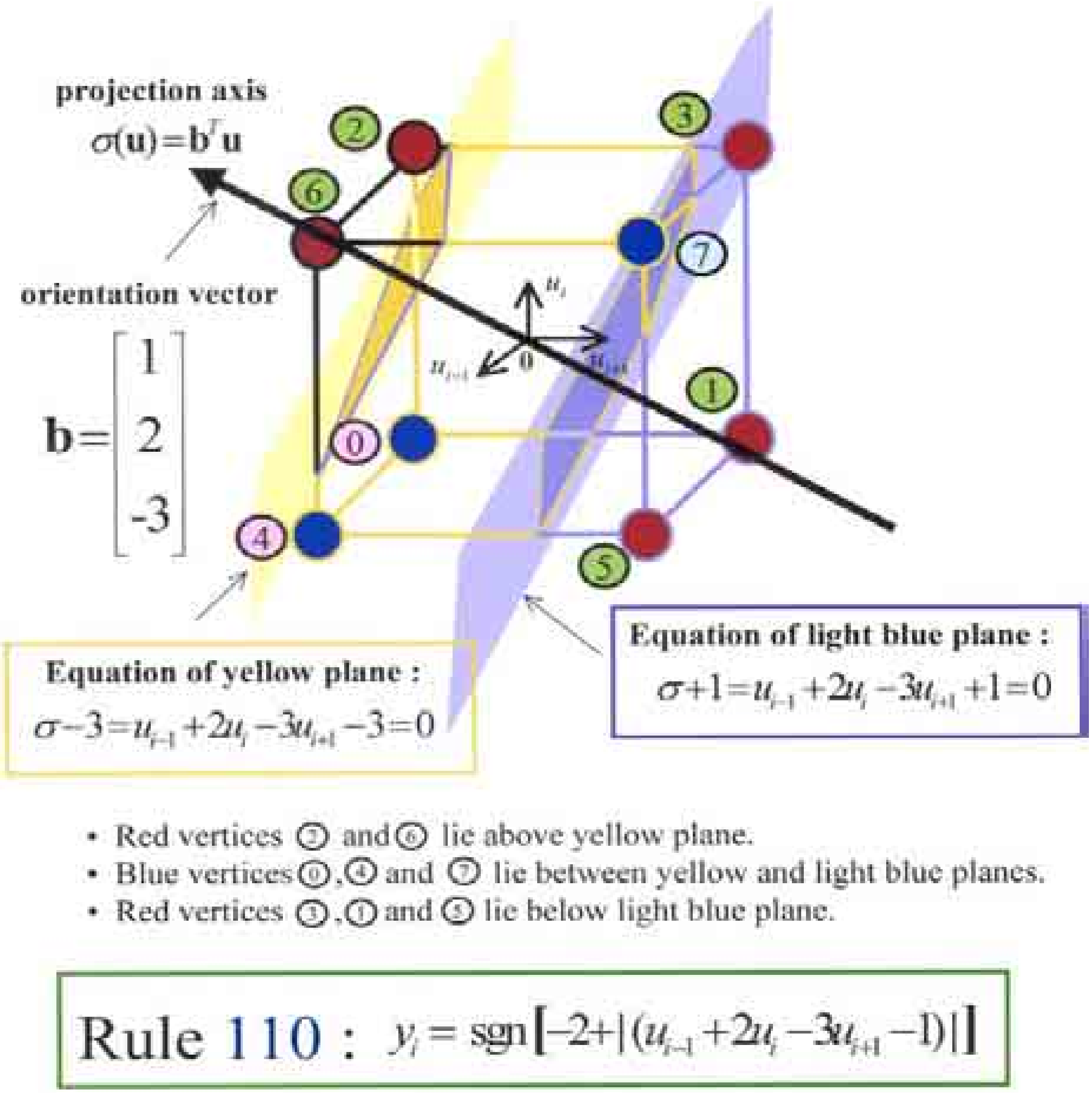


Fig. 8. Projection axis $\sigma(\mathbf{u})$ coincides with the orientation vector $\mathbf{b} = [1 \quad 2 \quad -3]^T$ drawn through the center "0" of the Boolean cube. It is orthogonal (i.e. intersects perpendicularly) to the two parallel planes.

have opted for *polynomials*. However, all rules in Table 2 with $\kappa = 3$ (integer in the upper right hand corner) would now require a third degree polynomial which would eat up significantly more space.

In a precise sense to be articulated below, every local Rule has a characteristic structure which can be identified by its separation planes. To stress the fundamental role played by the choice of separation planes in this local Rule *structural identification* process, we have extracted only the relevant parts of Fig. 8 and redrawn it in Fig. 9.

5.4. *A Local Rule with Three Separation Planes*

Some local Rules cannot be characterized by only two separation planes because it may not be possible to separate the eight vertices neatly into three colored groups and at the same time separate them by two *parallel* planes, no matter how we position the planes, or equivalently, no matter how we pick the *orientation* vector $\mathbf{b}$ (recall the separation planes are perpendicular to the orientation vector).

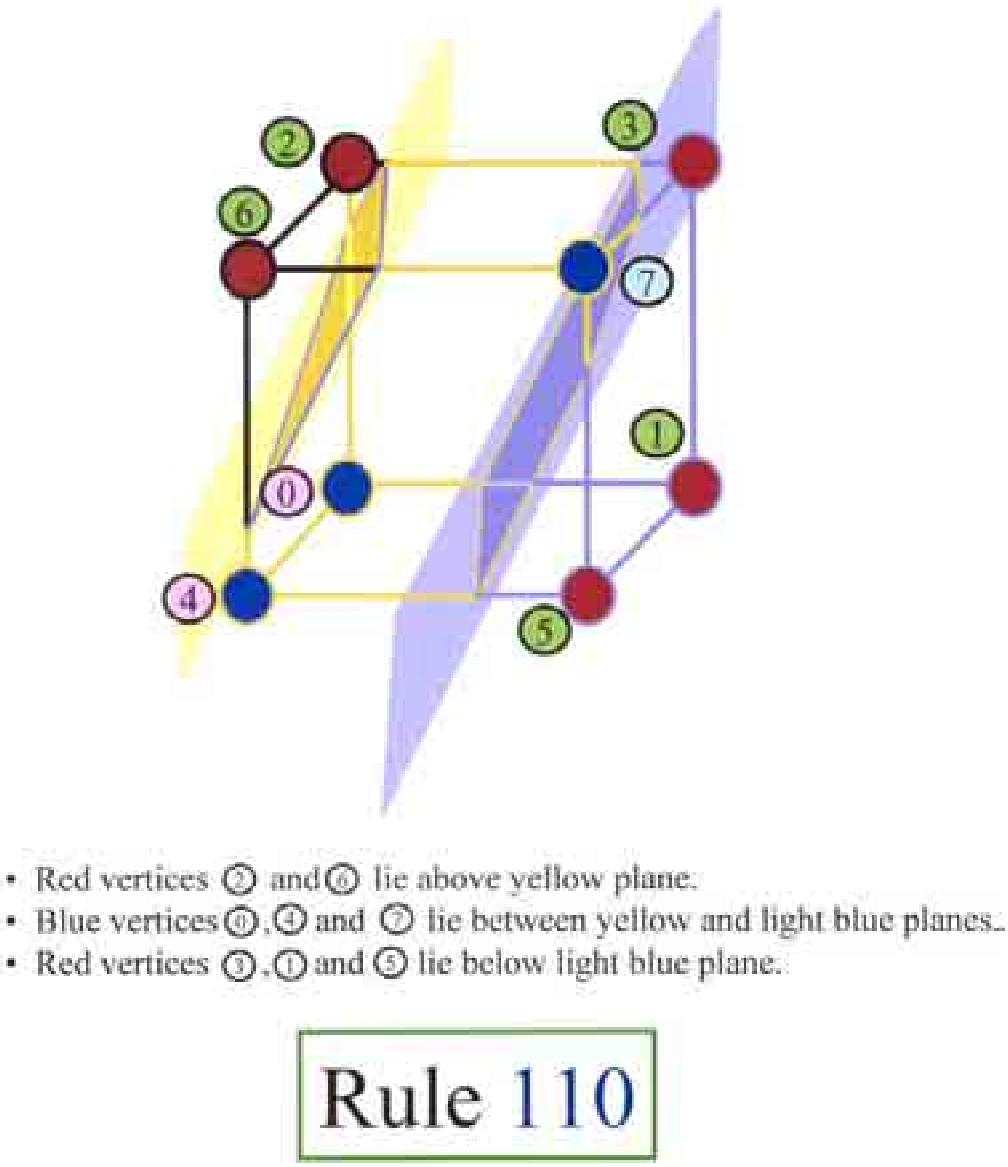


Fig. 9. Geometric structure of Rule 110.

However, it is always possible to choose a sufficiently large number of parallel planes to separate the vertices by simply projecting them onto a *projection* axis corresponding to almost[14] *every orientation vector* **b**. By scanning the sequence of red and blue crosses on the σ-axis, as in Fig. 7, we can always choose a sufficiently large number of *transition points* that would separate any number of groups of red crosses from neighboring groups of blue crosses. Once again, the key to this remarkably easy task is the *one-dimensional* character of the *projection* σ.

An example of a local Rule which cannot be separated by only two parallel planes is $\boxed{\text{Rule 150}}$. Suppose we pick the *orientation vector*, say $\mathbf{b} = [-4 \quad -2 \quad 1]^T$ and determine the corresponding projection axis $\sigma(\mathbf{u}) = -4u_{i-1} - 2u_i + u_{i+1}$ (pointing in a South-Eastern direction). Using the calculated data given in Table 5, we sketch the crosses along the normalized projection axis $\overline{\sigma}$ in Fig. 10(a). The redrawn unscaled version shown in Fig. 10(b) shows that we need five transition points to separate the crosses; namely, $\sigma_1 = -6$, $\sigma_2 = -2$, $\sigma_3 = 0$, $\sigma_4 = 2$,

[14]The only exception is when two vertices having different colors happen to project onto the same point on the σ-axis. This is a pathological situation that rarely occurs [Dogaru & Chua, 1999].

Table 5. Calculation of $y_i = \text{sgn}[w(\sigma)]$ for Rule 150 at vertex ⓝ with $\mathbf{b} = [-4 \quad -2 \quad 1]^T$.

n	u_{i-1}	u_i	u_{i+1}	$\bar{\sigma}(n)$	$\sigma(n)$	$w(n)$	y_i
0	-1	-1	-1	0.24	5	-1	-1
1	-1	-1	1	0.33	7	1	1
2	-1	1	-1	0.05	1	1	1
3	-1	1	1	0.14	3	-1	-1
4	1	-1	-1	-0.14	-3	1	1
5	1	-1	1	-0.05	-1	-1	-1
6	1	1	-1	-0.33	-7	-1	-1
7	1	1	1	-0.24	-5	1	1

and $\sigma_5 = 6$. A discriminant function $w(\sigma)$ having these five zero crossings is shown in Fig. 10(b). This function is given on top of Fig. 10(b). It follows that a dynamical system for generating the local Rule 150 is as follows:

$$\begin{aligned}\dot{x}_i = g(x_i) &+ (-4u_{i-1} - 2u_i + u_{i+1}) \\ &- |(-4u_{i-1} - 2u_i + u_{i+1} + 4)| \\ &+ |(-4u_{i-1} - 2u_i + u_{i+1} + 1)| \\ &- |(-4u_{i-1} - 2u_i + u_{i+1} - 1)| \\ &+ |(-4u_{i-1} - 2u_i + u_{i+1} - 4)| \qquad (24)\end{aligned}$$

Another dynamical system which yields the same local Rule in terms of a *polynomial* discriminant function $w(\sigma) = \sigma(\sigma+6)(\sigma+2)(\sigma-2)(\sigma-6)$ would lead to an even messier state equation involving a third degree polynomial of three variables u_{i-1}, u_i, and u_{i+1}.

Actually we can do even better by choosing another orientation vector; namely $\mathbf{b} = [-4 \quad -2 \quad 4]^T$. Repeating the exercise with this orientation vector leads to the simpler discriminant function (see Table 6 for the calculated data): $w(\sigma) = \sigma - |\sigma + 4| + |\sigma - 4|$ involving only three turning points (see Figs. 11 and 12). The corresponding state equation is given by:

$$\begin{aligned}\dot{x}_i = g(x_i) &+ (-4u_{i-1} - 2u_i + 4u_{i+1}) \\ &- |(-4u_{i-1} - 2u_i + 4u_{i+1} + 4)| \\ &+ |(-4u_{i-1} - 2u_i + 4u_{i+1} - 4)| \qquad (25)\end{aligned}$$

which is considerably simpler than Eq. (24).

In fact, by resorting to a composite structure of nested absolute value functions, it is always possible to transform Eq. (25) into the most compact form given in Eq. (5), namely,

State Equation for Rule 150

$$\dot{x}_i = g(x_i) + \{3 - |[7 - | - 3 + (-4u_{i-1} - 2u_i + 4u_{i+1})|]|\}$$

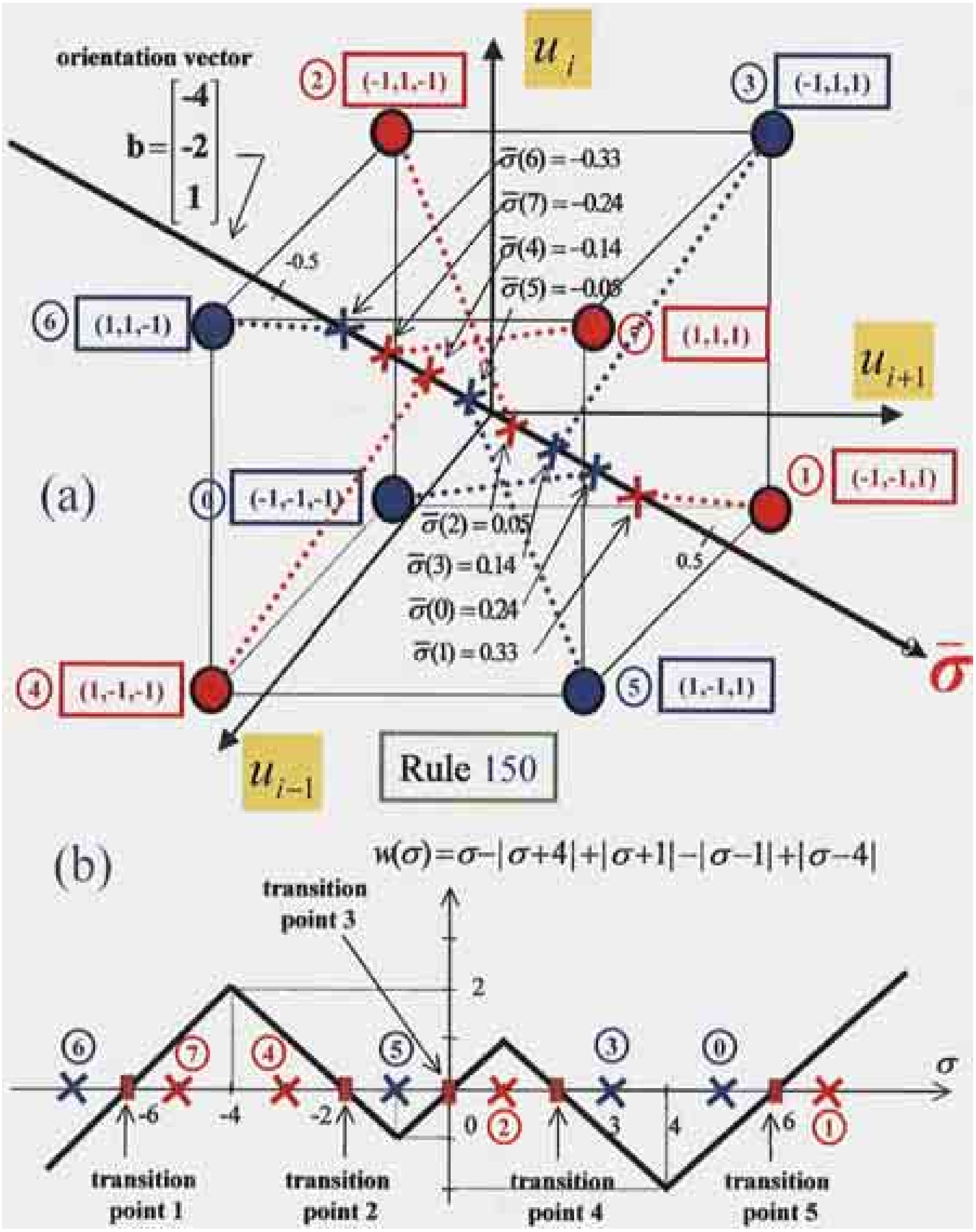


Fig. 10. (a) Projection of each vertex onto the normalized $\overline{\sigma}$-axis defined by $\mathbf{b} = [-4 \quad -2 \quad 1]^T$. (b) Actual projection $\sigma = \mathbf{b}^T\mathbf{u}$ of each vertex redrawn with separating curve $w(\sigma)$ shows five transition points, thereby requiring five parallel separating planes.

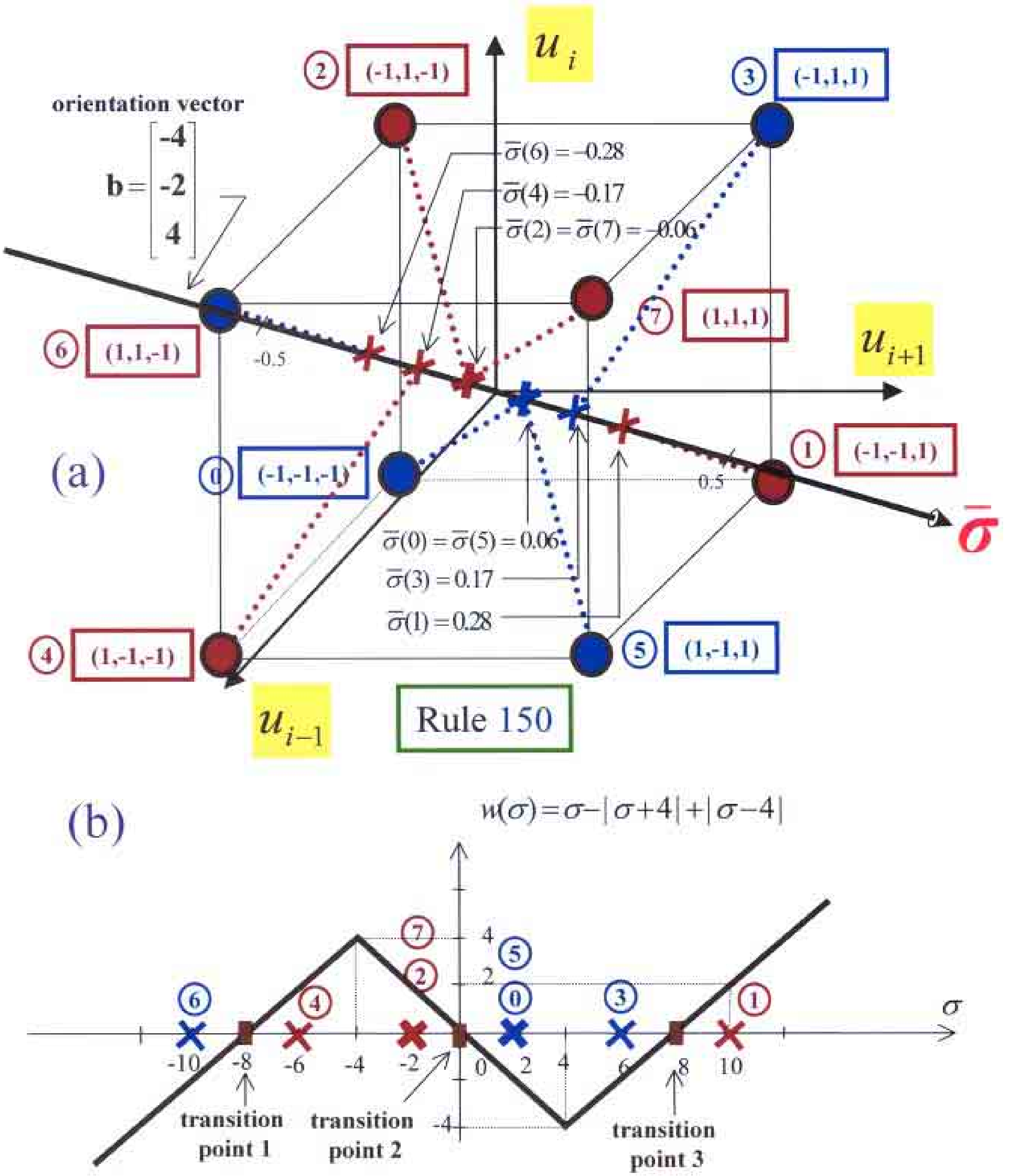


Fig. 11. (a) Projection of each vertex onto the normalized $\bar{\sigma}$-axis defined by $\mathbf{b} = [-4 \quad -2 \quad 4]^T$. (b) Actual projection $\sigma = \mathbf{b}^T\mathbf{u}$ of each vertex redrawn with separating curve $w(\sigma)$ shows three transition points, thereby requiring only three parallel separating planes.

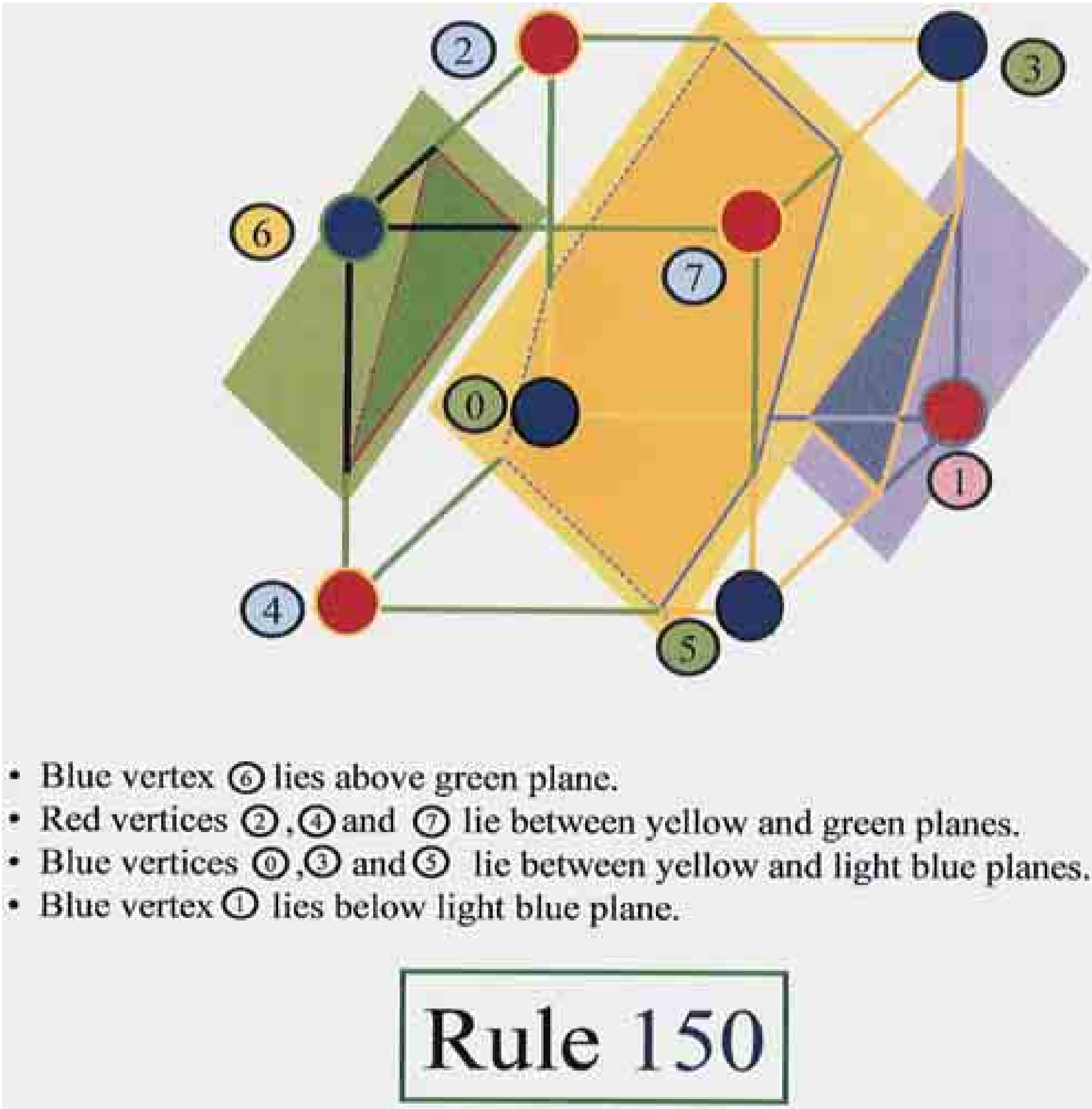


Fig. 12. Geometric structure of Rule 150.

Table 6. Calculation of $y_i = \text{sgn}[w(\sigma)]$ for Rule 150 at vertex ⓝ with $\mathbf{b} = [-4 \quad -2 \quad 4]^T$.

n	u_{i-1}	u_i	u_{i+1}	$\bar{\sigma}(n)$	$\sigma(n)$	$w(n)$	y_i
0	-1	-1	-1	0.06	2	-2	-1
1	-1	-1	1	0.28	10	2	1
2	-1	1	-1	-0.06	-2	2	1
3	-1	1	1	0.17	6	-2	-1
4	1	-1	-1	-0.17	-6	2	1
5	1	-1	1	0.06	2	-2	-1
6	1	1	-1	-0.28	-10	-2	-1
7	1	1	1	-0.06	-2	2	1

Table 7. Calculation of $y_i = \text{sgn}[w(\sigma)]$ for Rule 232 at vertex ⓝ with $\mathbf{b} = [1 \quad 1 \quad 1]^T$.

n	u_{i-1}	u_i	u_{i+1}	$\bar{\sigma}(n)$	$\sigma(n)$	$w(n)$	y_i
0	-1	-1	-1	-1	-3	-3	-1
1	-1	-1	1	-0.33	-1	-1	-1
2	-1	1	-1	-0.33	-1	-1	-1
3	-1	1	1	0.33	1	1	1
4	1	-1	-1	-0.33	-1	-1	-1
5	1	-1	1	0.33	1	1	1
6	1	1	-1	0.33	1	1	1
7	1	1	1	1	3	3	1

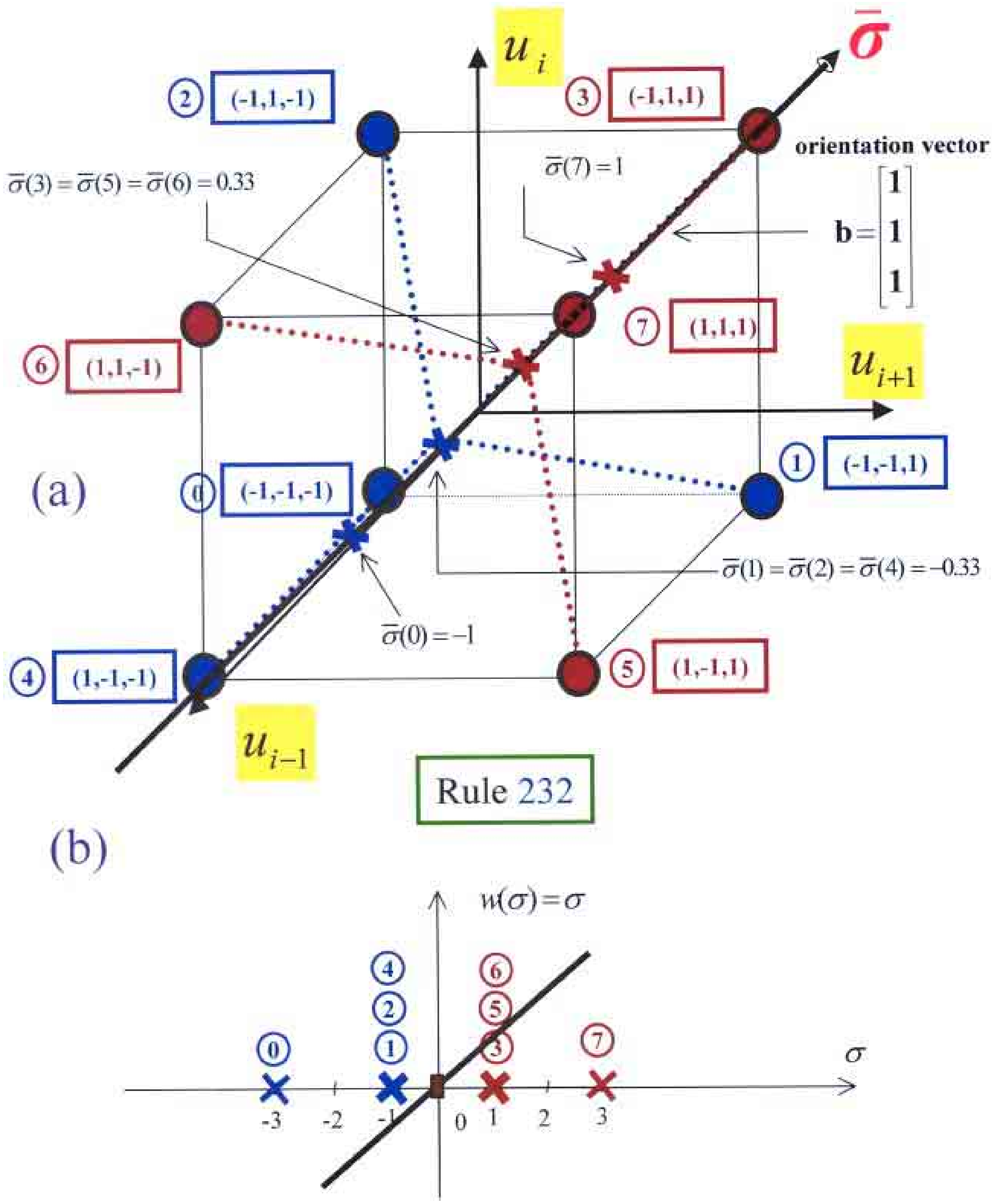


Fig. 13. (a) Projection of each vertex onto the normalized $\bar{\sigma}$-axis defined by $\mathbf{b} = [1 \quad 1 \quad 1]^T$. (b) Actual projection $\sigma = \mathbf{b}^T\mathbf{u}$ of each vertex redrawn with separating curve $w(\sigma)$ shows only one transition point, thereby requiring only one separating plane located at $\sigma = 0$, perpendicular to the orientation vector $\mathbf{b} = [1 \quad 1 \quad 1]^T$ in (a).

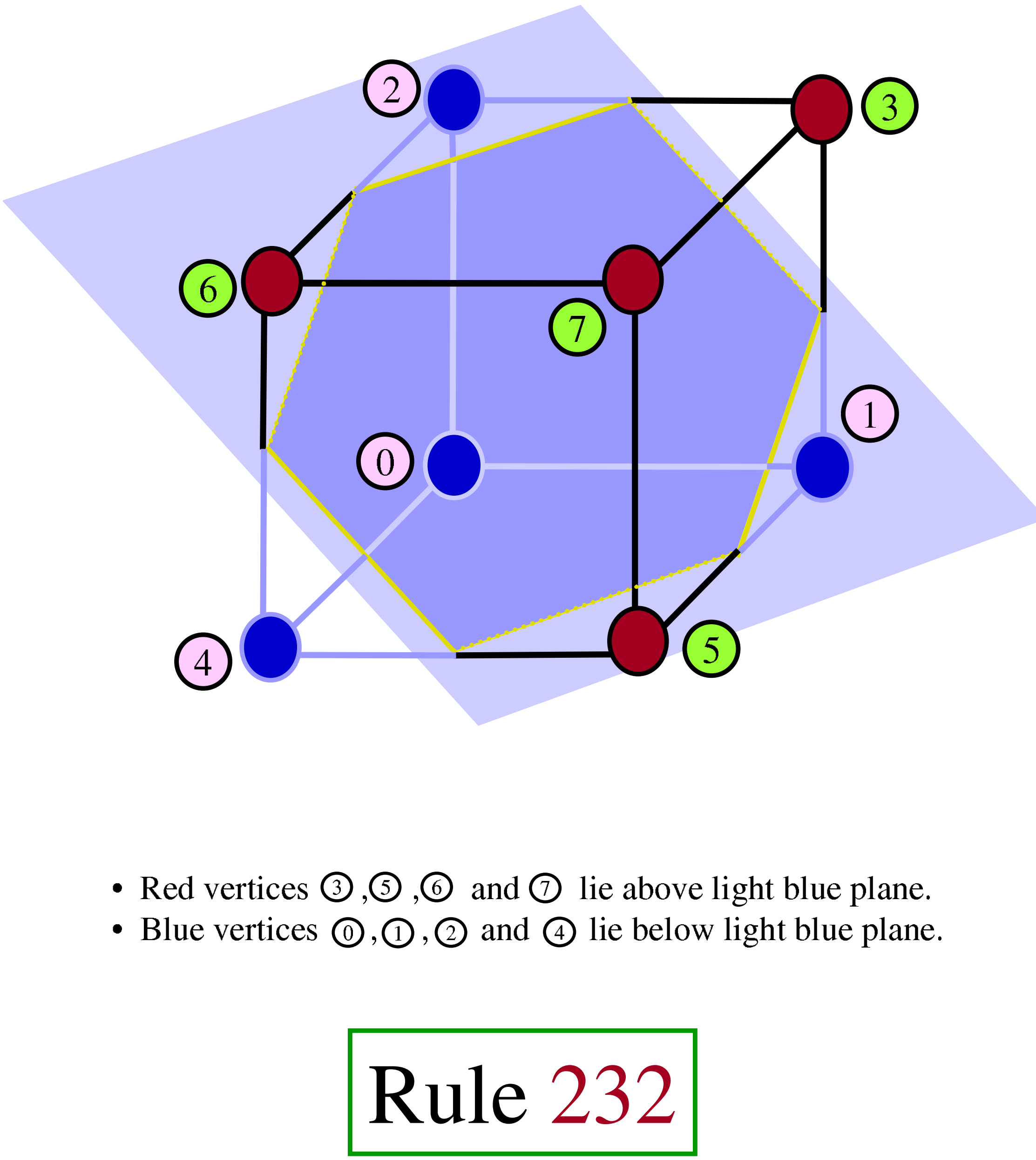


Fig. 14. Geometric structure of Rule 232.

It is this compact dynamical system which is listed in Table 2 for Rule 150.

5.5. *Linearly Separable Rules*

The colored vertices of many local Rules can be separated by only one plane. We will present two typical examples in the subsection.

Example 1. Rule 232

Using the orientation vector $\mathbf{b} = [1 \quad 1 \quad 1]^T$ and the data calculated for Rule 232 in Table 7, we obtain

Table 8. Calculation of $y_i = \text{sgn}[w(\sigma)]$ for Rule 250 at vertex ⓝ with $\mathbf{b} = [1 \quad 0 \quad 1]^T$.

n	u_{i-1}	u_i	u_{i+1}	$\bar{\sigma}(n)$	$\sigma(n)$	$w(n)$	y_i
0	-1	-1	-1	-1	-2	-1	-1
1	-1	-1	1	0	0	1	1
2	-1	1	-1	-1	-2	-1	-1
3	-1	1	1	0	0	1	1
4	1	-1	-1	0	0	1	1
5	1	-1	1	1	2	3	1
6	1	1	-1	0	0	1	1
7	1	1	1	1	2	3	1

the projections shown in Fig. 13. Since there is only *one transition point*, the colored vertices of Rule 232 can be separated by only one plane, as shown in Fig. 14.

Example 2. Rule 250

Using the orientation vector $\mathbf{b} = [1 \quad 0 \quad 1]^T$ and the data calculated for Rule 250 in Table 8, we obtain the projections shown in Fig. 15. Once again, there is only *one transition point* and hence the colored vertices of Rule 250 can also be separated by only one plane, as shown in Fig. 16.

Any local Rule whose colored vertices can be separated by only one plane is said to be *linearly separable* because its associated discriminant function $w(\sigma)$ is a *straight line*. A careful examination of the 256 Boolean cubes in Table 1 shows that 104 among them are linearly separable. These are the local Rules whose Rule number N is printed in *Red*. The remaining 152 Rule numbers which are printed in *blue* are *not* locally separable.

In addition to Rule 110 and Rule 150 presented earlier, a sample of five other *Linearly-Non-Separable* Rules cited in [Wolfram, 2002], among many others, are shown in Figs. 17–21; namely, Rule 20, Rule 22, Rule 30, Rule 90, and Rule 108.

For future reference, all 104 Linearly Separable Rules and all 152 Linearly-Non-Separable Rules are listed in Tables 9 and 10, respectively.

5.6. *Complexity Index*

By applying the *projection* technique illustrated in the preceding sections, it is clear that the colored vertices of every Boolean cube in Table 1 can be

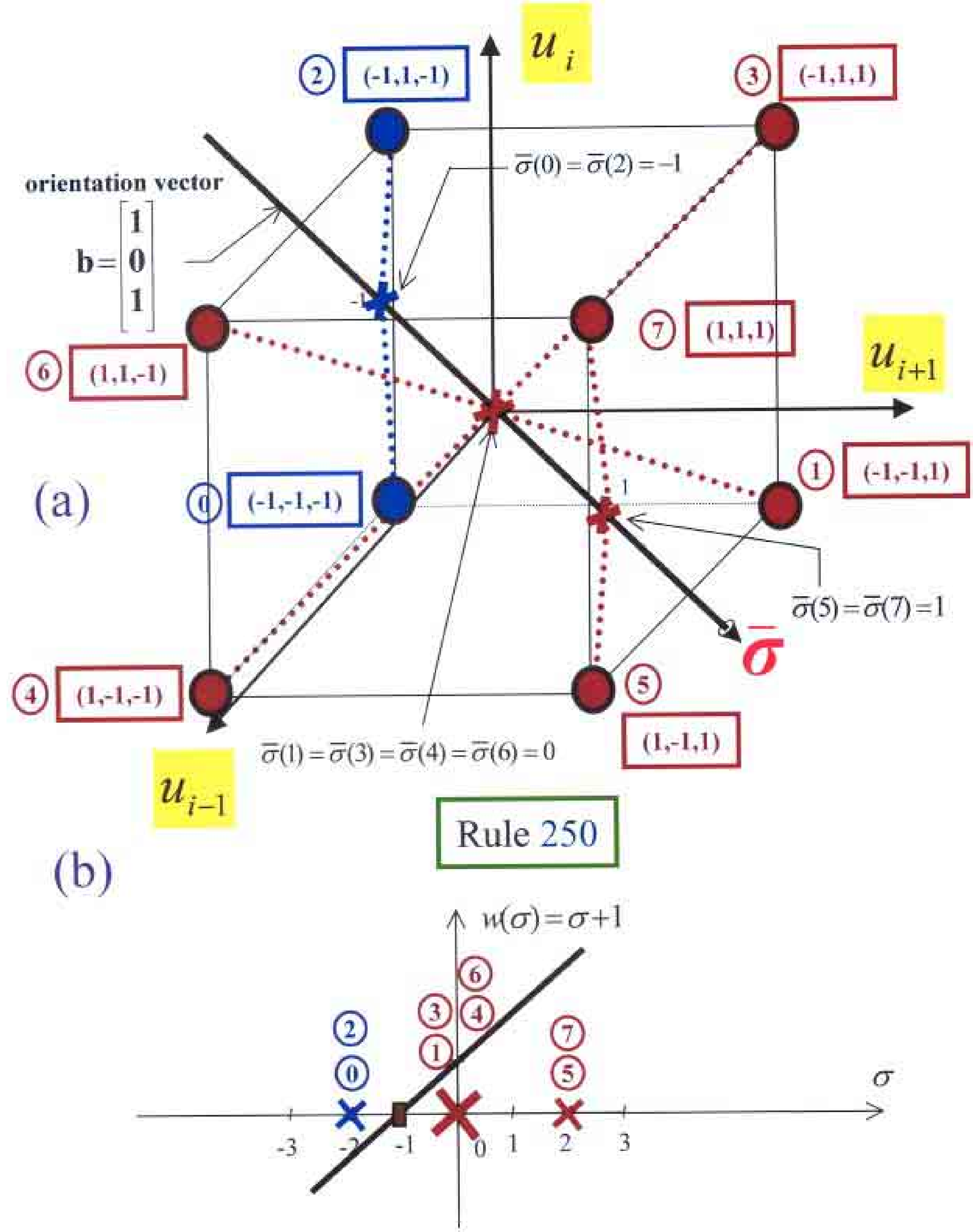


Fig. 15. (a) Projection of each vertex onto the normalized $\bar{\sigma}$-axis defined by $\mathbf{b} = [1 \quad 0 \quad 1]^T$. (b) Actual projection $\sigma = \mathbf{b}^T\mathbf{u}$ of each vertex redrawn with separating curve $w(\sigma)$ shows only one transition point, thereby requiring only one separating plane located at $\sigma = -1$, perpendicular to the orientation vector $\mathbf{b} = [1 \quad 0 \quad 1]^T$ in (a).

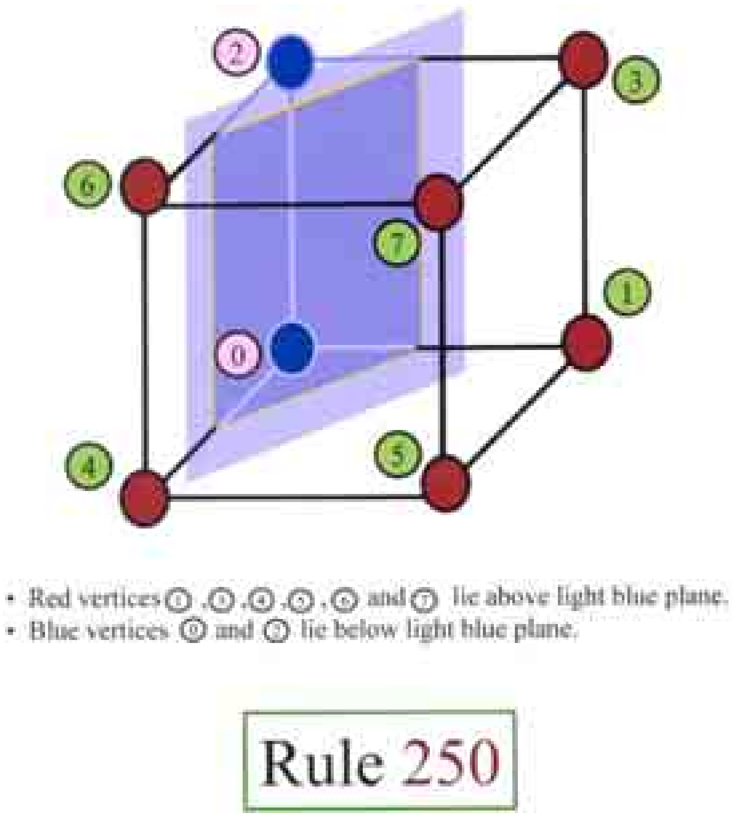


Fig. 16. Geometric structure of Rule 250.

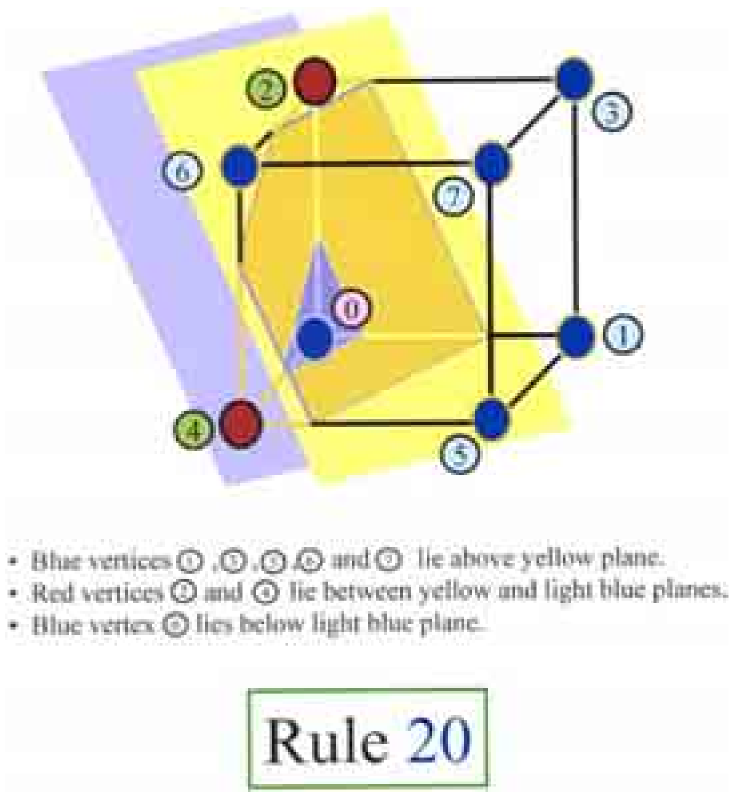


Fig. 17. Geometric structure of Rule 20.

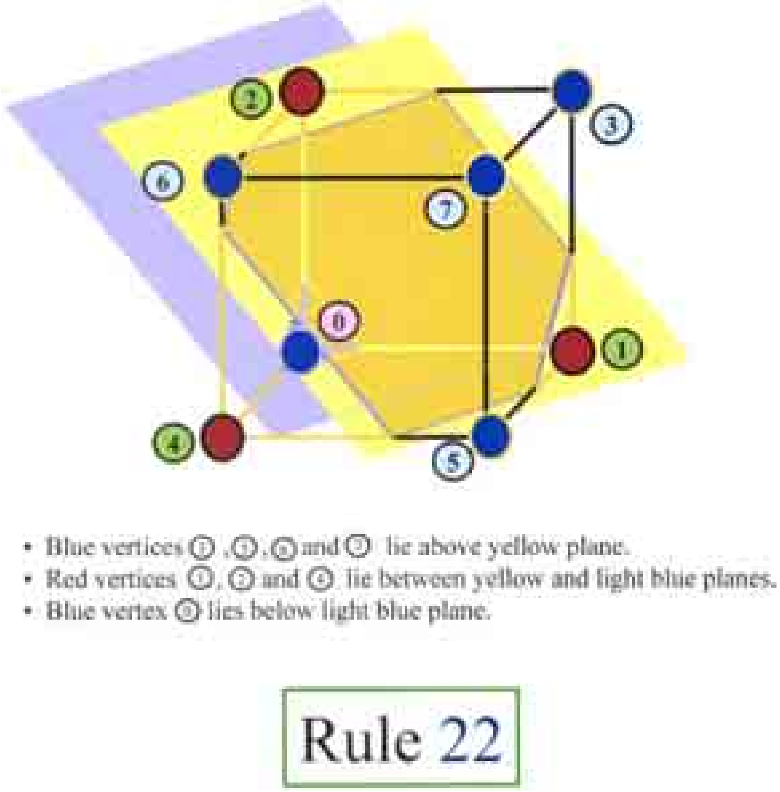


Fig. 18. Geometric structure of Rule 22.

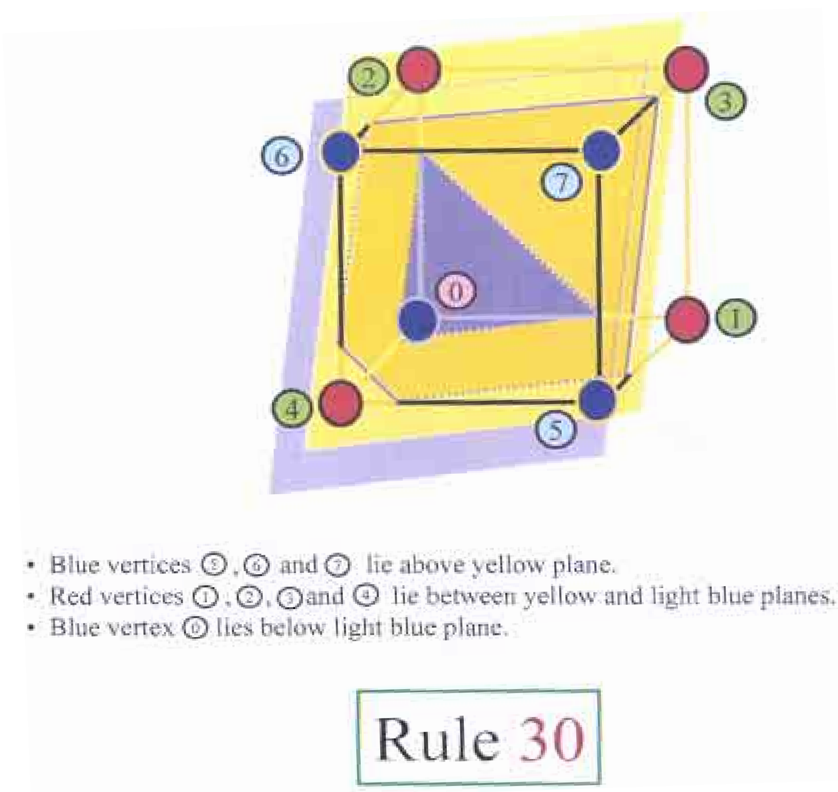


Fig. 19. Geometric structure of Rule 30.

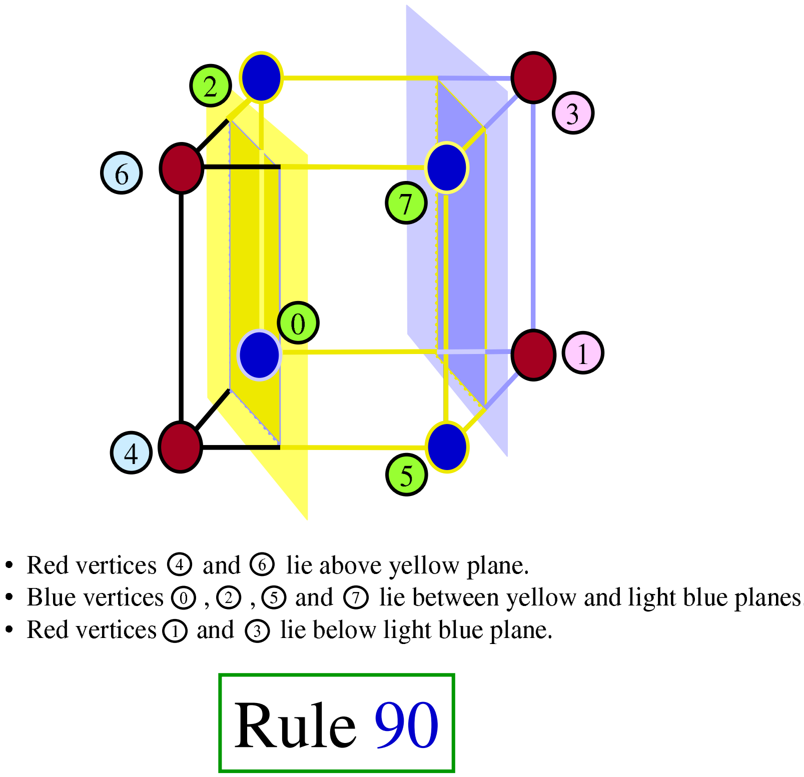


Fig. 20. Geometric structure of Rule 90.

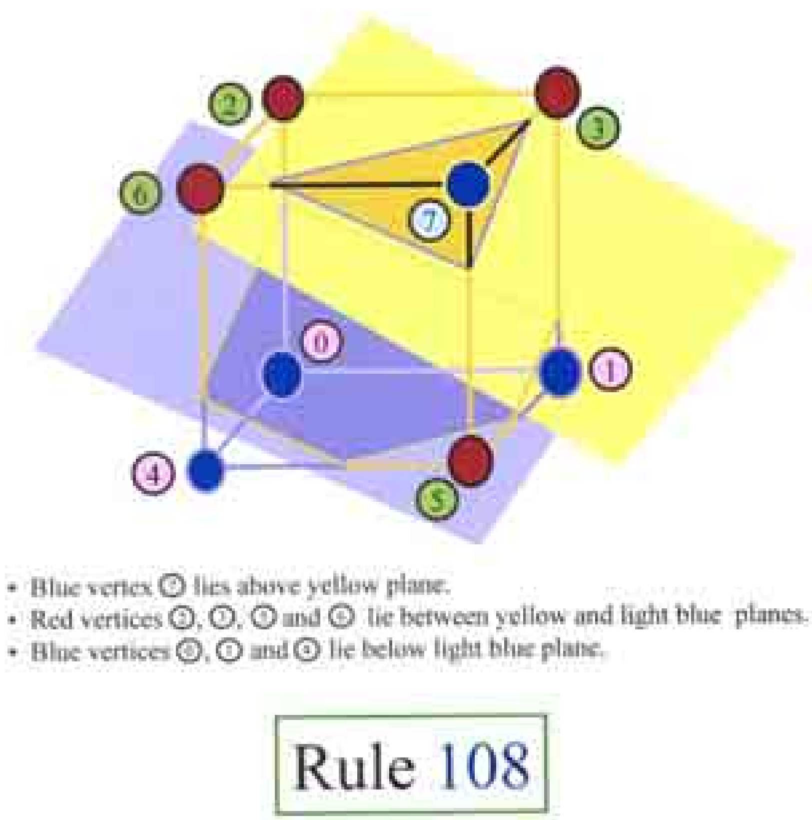


Fig. 21. Geometric structure of Rule 108.

Table 9. List of 104 Linearly Separable Boolean Function Rules.

0	1	2	3	4	5	7	8	10	11	12	13	14
15	16	17	19	21	23	31	32	34	35	42	43	47
48	49	50	51	55	59	63	64	68	69	76	77	79
80	81	84	85	87	93	95	112	113	115	117	119	127
128	136	138	140	142	143	160	162	168	170	171	174	175
176	178	179	186	187	191	192	196	200	204	205	206	207
208	212	213	220	221	223	224	232	234	236	238	239	240
241	242	243	244	245	247	248	250	251	252	253	254	255

Table 10. List of 152 Linearly Non-Separable Boolean Function Rules.

6	9	18	20	22	24	25	26	27	28	29	30	33
36	37	38	39	40	41	44	45	46	52	53	54	56
57	58	60	61	62	65	66	67	70	71	72	73	74
75	78	82	83	86	88	89	90	91	92	94	96	97
98	99	100	101	102	103	104	105	106	107	108	109	110
111	114	116	118	120	121	122	123	124	125	126	129	130
131	132	133	134	135	137	139	141	144	145	146	147	148
149	150	151	152	153	154	155	156	157	158	159	161	163
164	165	166	167	169	172	173	177	180	181	182	183	184
185	188	189	190	193	194	195	197	198	199	201	202	203
209	210	211	214	215	216	217	218	219	222	225	226	227
228	229	230	231	233	235	237	246	249				

Table 11. Complexity Index κ of Local Rules.

$\kappa = 1$	$\kappa = 2$	$\kappa = 3$			
All linearly-Separable rules (Table 9)	All linearly Non-Separable rules (Table 10) that do not appear on the right	27	228	83	172
		29	226	71	184
		39	216	53	202
		46	209	116	139
		58	197	114	141
		78	177	92	163
		105	150		

separated by a finite number of parallel planes. In general, each local rule can be separated by various numbers of parallel planes, each one giving rise to a different state equation which codes for the same Rule. In other words, many *distinct dynamical systems can be used to code for any local Rule* listed in Table 1. The 256 dynamical systems listed in Table 2 represent only one choice with a compact formula.

However, there is a *unique integer* κ, henceforth called the *complexity index* of a local Rule, which characterizes the geometrical structure of the corresponding Boolean cube, namely the *minimum* number of *parallel planes* that is *necessary* to separate the colored vertices. Hence, *all linearly separable* Rules have a complexity index of $\kappa = 1$. A careful analysis of Table 1 shows that each of the remaining 152 Linearly Non-Separable Rules has a complexity index of either 2 or 3 (see Table 11). For example, Rule 110 has a complexity index of $\kappa = 2$ whereas Rule 150 has a complexity of index $\kappa = 3$. The complexity index κ of each local Rule is printed in the upper right-hand corner of each quadrant of Table 2.

5.7. *Every Local Rule is a Member of an Equivalence Class*

Two local Rules N_1 and N_2 are said to be *equivalent* iff there exists a transformation which maps Rule N_1 onto Rule N_2, and vice versa. The following are two useful symmetry transformations:

1. Red $\leftrightarrow$ Blue complementary transformation

The complement of a local Rule N_1 is a local Rule N_2 where the colors of corresponding vertices of corresponding Boolean cubes from Table 1 are *complement* of each other i.e. corresponding *red* vertices become *blue*, and vice versa. Since the dynamics of Local Rule N_1 can be predicted from that of Local Rule N_2 and vice-versa, we say Rules N_1 and N_2 form a "Red $\leftrightarrow$ Blue complementary pair". Clearly, half of the Local Rules form "Red $\leftrightarrow$ Blue complementary pairs" with the other half. Table 12 gives the Red $\leftrightarrow$ Blue complementary pair of all Local Rules from $N^* = 128$ to $N^* = 255$.

2. Left $\leftrightarrow$ Right symmetrical transformation

The Left $\leftrightarrow$ Right symmetrical transformation of a local Rule N_1 is a local Rule N_2 obtained by interchanging the colors between vertices ③ and ⑥, as well as between vertices ① and ④ in the Boolean cube N_1 from Table 1. Since the left and right neighbors of each cell belonging to a left–right symmetrical pair in Table 2 will move in opposite but laterally symmetric directions, we can predict the

Table 12. List of 128 Red ↔ Blue complementary pairs of Boolean Function Rules.

N	N*	N	N*	N	N*	N	N*
127	128	95	160	63	192	31	224
126	129	94	161	62	193	30	225
125	130	93	162	61	194	29	226
124	131	92	163	60	195	28	227
123	132	91	164	59	196	27	228
122	133	90	165	58	197	26	229
121	134	89	166	57	198	25	230
120	135	88	167	56	199	24	231
119	136	87	168	55	200	23	232
118	137	86	169	54	201	22	233
117	138	85	170	53	202	21	234
116	139	84	171	52	203	20	235
115	140	83	172	51	204	19	236
114	141	82	173	50	205	18	237
113	142	81	174	49	206	17	238
112	143	80	175	48	207	16	239
111	144	79	176	47	208	15	240
110	145	78	177	46	209	14	241
109	146	77	178	45	210	13	242
108	147	76	179	44	211	12	243
107	148	75	180	43	212	11	244
106	149	74	181	42	213	10	245
105	150	73	182	41	214	9	246
104	151	72	183	40	215	8	247
103	152	71	184	39	216	7	248
102	153	70	185	38	217	6	249
101	154	69	186	37	218	5	250
100	155	68	187	36	219	4	251
99	156	67	188	35	220	3	252
98	157	66	189	34	221	2	253
97	158	65	190	33	222	1	254
96	159	64	191	32	223	0	255

motion of one local Rule from the motion of its left–right symmetric pair, and vice versa. For example, Rules 110 and 124 form a left–right symmetrical pair because the color (blue) of vertex ④ in the Boolean cube 110 in Table 1 becomes the color of vertex ① of Boolean cube 124, and the color (red) of vertex ① becomes the color of vertex ④. The color of all other vertices are identical. Note that in this case, the colors of vertices ③ and ⑥ remain unchanged because they are identical (both are red) so

Table 13. List of 96 left ↔ right symmetrical pairs of Boolean Function Rules.

$\overleftarrow{N}$	$\overrightarrow{N}$	$\overleftarrow{N}$	$\overrightarrow{N}$	$\overleftarrow{N}$	$\overrightarrow{N}$	$\overleftarrow{N}$	$\overrightarrow{N}$
2	16	40	96	130	144	168	224
3	17	41	97	131	145	169	225
6	20	42	112	134	148	170	240
7	21	43	113	135	149	171	241
8	64	44	100	136	192	172	228
9	65	45	101	137	193	173	229
10	80	46	116	138	208	174	244
11	81	47	117	139	209	175	245
12	68	56	98	140	196	184	226
13	69	57	99	141	197	185	227
14	84	58	114	142	212	186	242
15	85	59	115	143	213	187	243
24	66	60	102	152	194	188	230
25	67	61	103	153	195	189	231
26	82	62	118	154	210	190	246
27	83	63	119	155	211	191	247
28	70	74	88	156	198	202	216
29	71	75	89	157	199	203	217
30	86	78	92	158	214	206	220
31	87	79	93	159	215	207	221
34	48	106	120	162	176	234	248
35	49	107	121	163	177	235	249
38	52	110	124	166	180	238	252
39	53	111	125	167	181	239	253

that interchanging them leads to the same pattern. Observe next that the local Rule 150 is *invariant* under a left ↔ right symmetrical transformation because vertices ① and ④ have identical colors (both are red); similarly, vertices ③ and ⑥ also have identical colors (both are blue). Table 13 gives the complete list of 96 distinct left ↔ right symmetrical pairs.

Combining Tables 12 and 13, we obtain Table 14 which lists the complete set of equivalent classes of all local Rules from Table 1. The shaded entries in this table correspond to those local Rules that remain *invariant* under a left–right symmetrical transformation, such as Rule 150.

Since all members belonging to the same equivalent class of local Rules have identical complexity indices, and exhibit dynamic behaviors that can be predicted from each other, it suffices to undertake an in-depth analysis of only one member from each equivalent class. Using Tables 9, 10 and 14, we have identified 33 *independent Linearly Separable* local Rules (see Table 15) and 47 *independent Linearly*

Table 14. Equivalent Classes of Boolean Function Rules.

Boolean Rule Number	Red ↔ Blue Vertex transformation	Left ↔ Right Vertex transformation
0	255	
1	254	
2	253	16
3	252	17
4	251	
5	250	
6	249	20
7	248	21
8	247	64
9	246	65
10	245	80
11	244	81
12	243	68
13	242	69
14	241	84
15	240	85
16	239	2
17	238	3
18	237	
19	236	
20	235	6
21	234	7
22	233	
23	232	
24	231	66
25	230	67
26	229	82
27	228	83
28	227	70
29	226	71
30	225	86
31	224	87
32	223	
33	222	
34	221	48
35	220	49
36	219	
37	218	
38	217	52
39	216	53
40	215	96
41	214	97
42	213	112

Boolean Rule Number	Red ↔ Blue Vertex transformation	Left ↔ Right Vertex transformation
43	212	113
44	211	100
45	210	101
46	209	116
47	208	117
48	207	34
49	206	35
50	205	
51	204	
52	203	38
53	202	39
54	201	
55	200	
56	199	98
57	198	99
58	197	114
59	196	115
60	195	102
61	194	103
62	193	118
63	192	119
64	191	8
65	190	9
66	189	24
67	188	25
68	187	12
69	186	13
70	185	28
71	184	29
72	183	
73	182	
74	181	88
75	180	89
76	179	
77	178	
78	177	92
79	176	93
80	175	10
81	174	11
82	173	26
83	172	27
84	171	14
85	170	15

Boolean Rule Number	Red ↔ Blue Vertex Transformation	Left ↔ Right Vertex transformation
86	169	30
87	168	31
88	167	74
89	166	75
90	165	
91	164	
92	163	78
93	162	79
94	161	
95	160	
96	159	40
97	158	41
98	157	56
99	156	57
100	155	44
101	154	45
102	153	60
103	152	61
104	151	
105	150	
106	149	120
107	148	121
108	147	
109	146	
110	145	124
111	144	125
112	143	42
113	142	43
114	141	58
115	140	59
116	139	46
117	138	47
118	137	62
119	136	63
120	135	106
121	134	107
122	133	
123	132	
124	131	110
125	130	111
126	129	
127	128	
128	127	

Table 14. (*Continued*)

Boolean Rule Number	Red ↔ Blue Vertex Transformation	Left ↔ Right Vertex transformation
129	126	
130	125	144
131	124	145
132	123	
133	122	
134	121	148
135	120	149
136	119	192
137	118	193
138	117	208
139	116	209
140	115	196
141	114	197
142	113	212
143	112	213
144	111	130
145	110	131
146	109	
147	108	
148	107	134
149	106	135
150	105	
151	104	
152	103	194
153	102	195
154	101	210
155	100	211
156	99	198
157	98	199
158	97	214
159	96	215
160	95	
161	94	
162	93	176
163	92	177
164	91	
165	90	
166	89	180
167	88	181
168	87	224
169	86	225
170	85	240
171	84	241

Boolean Rule Number	Red ↔ Blue Vertex Transformation	Left ↔ Right Vertex transformation
172	83	228
173	82	229
174	81	244
175	80	245
176	79	162
177	78	163
178	77	
179	76	
180	75	166
181	74	167
182	73	
183	72	
184	71	226
185	70	227
186	69	242
187	68	243
188	67	230
189	66	231
190	65	246
191	64	247
192	63	136
193	62	137
194	61	152
195	60	153
196	59	140
197	58	141
198	57	156
199	56	157
200	55	
201	54	
202	53	216
203	52	217
204	51	
205	50	
206	49	220
207	48	221
208	47	138
209	46	139
210	45	154
211	44	155
212	43	142
213	42	143
214	41	158

Boolean Rule Number	Red ↔ Blue Vertex Transformation	Left ↔ Right Vertex transformation
215	40	159
216	39	202
217	38	203
218	37	
219	36	
220	35	206
221	34	207
222	33	
223	32	
224	31	168
225	30	169
226	29	184
227	28	185
228	27	172
229	26	229
230	25	188
231	24	189
232	23	
233	22	
234	21	248
235	20	249
236	19	
237	18	
238	17	252
239	16	253
240	15	170
241	14	171
242	13	186
243	12	187
244	11	174
245	10	175
246	9	190
247	8	191
248	7	234
249	6	235
250	5	
251	4	
252	3	238
253	2	239
254	1	
255	0	

Table 15. List of 33 independent Linearly Separable Boolean Function Rules.

0	1	2	3	4	5	7	8	10	11	12	13
14	15	19	23	31	32	34	35	42	43	47	50
51	55	59	63	76	77	79	95	127			

Table 16. List of 47 independent Linearly Non-Separable Boolean Function Rules.

6	9	18	22	24	25	26	27	28	29	30	33
36	37	38	39	40	41	44	45	46	54	56	57
58	60	61	62	72	73	74	75	78	90	91	94
104	105	106	107	108	109	110	111	122	123	126	

Table 17. Synthesis of Linearly Non-Separable Boolean Function Rules via Elementary (AND, OR) Boolean Operation on Linearly-Separable Rules.

	Non-separable Boolean Function Rule N	Linearly-separable Boolean Function Rules			
1	Rule 6	Rule 7	AND	Rule 14	
2	Rule 9	Rule 11	AND	Rule 13	
3	Rule 18	Rule 19	AND	Rule 50	
4	Rule 20	Rule 21	AND	Rule 84	
5	Rule 22	Rule 23	AND	Rule 254	
6	Rule 24	Rule 31	AND	Rule 248	
7	Rule 25	Rule 59	AND	Rule 221	
8	Rule 26	Rule 31	AND	Rule 250	
9	Rule 27	Rule 31	AND	Rule 251	
10	Rule 28	Rule 31	AND	Rule 220	
11	Rule 29	Rule 31	AND	Rule 93	
12	Rule 30	Rule 31	AND	Rule 254	
13	Rule 33	Rule 35	AND	Rule 49	
14	Rule 36	Rule 47	AND	Rule 244	
15	Rule 37	Rule 47	AND	Rule 117	
16	Rule 38	Rule 55	AND	Rule 238	
17	Rule 39	Rule 47	AND	Rule 55	
18	Rule 40	Rule 42	AND	Rule 168	
19	Rule 41	Rule 43	AND	Rule 253	
20	Rule 44	Rule 47	AND	Rule 236	
21	Rule 45	Rule 47	AND	Rule 253	
22	Rule 46	Rule 47	AND	Rule 254	
23	Rule 52	Rule 55	AND	Rule 244	
24	Rule 53	Rule 55	AND	Rule 245	
25	Rule 54	Rule 55	AND	Rule 254	
26	Rule 56	Rule 59	AND	Rule 248	

	Non-separable Boolean Function Rule N	Linearly-separable Boolean Function Rules			
27	Rule 57	Rule 59	AND	Rule 253	
28	Rule 58	Rule 59	AND	Rule 254	
29	Rule 60	Rule 63	AND	Rule 252	
30	Rule 61	Rule 63	AND	Rule 253	
31	Rule 62	Rule 63	AND	Rule 254	
32	Rule 65	Rule 69	AND	Rule 81	
33	Rule 66	Rule 79	AND	Rule 242	
34	Rule 67	Rule 79	AND	Rule 243	
35	Rule 70	Rule 87	AND	Rule 206	
36	Rule 71	Rule 79	AND	Rule 87	
37	Rule 72	Rule 76	AND	Rule 232	
38	Rule 73	Rule 77	AND	Rule 251	
39	Rule 74	Rule 79	AND	Rule 234	
40	Rule 75	Rule 79	AND	Rule 251	
41	Rule 78	Rule 79	AND	Rule 254	
42	Rule 82	Rule 87	AND	Rule 250	
43	Rule 83	Rule 87	AND	Rule 243	
44	Rule 86	Rule 87	AND	Rule 254	
45	Rule 88	Rule 93	AND	Rule 248	
46	Rule 89	Rule 93	AND	Rule 251	
47	Rule 90	Rule 95	AND	Rule 250	
48	Rule 91	Rule 95	AND	Rule 251	
49	Rule 92	Rule 95	AND	Rule 220	
50	Rule 94	Rule 95	AND	Rule 254	
51	Rule 96	Rule 112	AND	Rule 224	
52	Rule 97	Rule 113	AND	Rule 239	

	Non-separable Boolean Function Rule N	Linearly-separable Boolean Function Rules			
53	Rule 98	Rule 115	AND	Rule 234	
54	Rule 99	Rule 115	AND	Rule 239	
55	Rule 100	Rule 117	AND	Rule 238	
56	Rule 101	Rule 117	AND	Rule 239	
57	Rule 102	Rule 119	AND	Rule 238	
58	Rule 103	Rule 119	AND	Rule 239	
59	Rule 104	Rule 127	AND	Rule 232	
60	Rule 105	(Rule 43	OR	Rule 64)	AND Rule 253
61	Rule 106	Rule 127	AND	Rule 234	
62	Rule 107	Rule 43	OR	Rule 64	
63	Rule 108	Rule 127	AND	Rule 236	
64	Rule 109	Rule 77	OR	Rule 32	
65	Rule 110	Rule 127	AND	Rule 238	
66	Rule 111	Rule 127	AND	Rule 239	
67	Rule 114	Rule 115	AND	Rule 242	
68	Rule 116	Rule 117	AND	Rule 252	
69	Rule 118	Rule 119	AND	Rule 254	
70	Rule 120	Rule 127	AND	Rule 248	
71	Rule 121	Rule 113	OR	Rule 8	
72	Rule 122	Rule 127	AND	Rule 250	
73	Rule 123	Rule 127	AND	Rule 251	
74	Rule 124	Rule 127	AND	Rule 252	
75	Rule 125	Rule 127	AND	Rule 253	
76	Rule 126	Rule 127	AND	Rule 254	
77	Rule 129	Rule 143	AND	Rule 241	
78	Rule 130	Rule 138	AND	Rule 162	

Table 17. (*Continued*)

	Non-separable Boolean Function Rule N	Linearly-separable Boolean Function Rules			
79	Rule 131	Rule 143	AND	Rule 243	
80	Rule 132	Rule 140	AND	Rule 196	
81	Rule 133	Rule 143	AND	Rule 245	
82	Rule 134	Rule 142	AND	Rule 247	
83	Rule 135	Rule 143	AND	Rule 247	
84	Rule 137	Rule 171	AND	Rule 221	
85	Rule 139	Rule 143	AND	Rule 251	
86	Rule 141	Rule 143	AND	Rule 205	
87	Rule 144	Rule 176	AND	Rule 213	
88	Rule 145	Rule 179	AND	Rule 213	
89	Rule 146	Rule 178	AND	Rule 223	
90	Rule 147	Rule 179	AND	Rule 223	
91	Rule 148	Rule 191	AND	Rule 212	
92	Rule 149	Rule 191	AND	Rule 213	
93	Rule 150	(Rule 23	OR	Rule 128)	AND Rule 254
94	Rule 151	Rule 128	OR	Rule 23	
95	Rule 152	Rule 186	AND	Rule 220	
96	Rule 153	Rule 187	AND	Rule 221	
97	Rule 154	Rule 186	AND	Rule 223	
98	Rule 155	Rule 187	AND	Rule 223	
99	Rule 156	Rule 191	AND	Rule 220	
100	Rule 157	Rule 191	AND	Rule 221	
101	Rule 158	Rule 142	OR	Rule 16	
102	Rule 159	Rule 191	AND	Rule 223	
103	Rule 161	Rule 171	AND	Rule 241	
104	Rule 163	Rule 171	AND	Rule 179	
105	Rule 164	Rule 174	AND	Rule 244	
106	Rule 165	Rule 175	AND	Rule 245	
107	Rule 166	Rule 174	AND	Rule 247	
108	Rule 167	Rule 175	AND	Rule 247	
109	Rule 169	Rule 171	AND	Rule 253	
110	Rule 172	Rule 174	AND	Rule 253	
111	Rule 173	Rule 175	AND	Rule 253	
112	Rule 177	Rule 179	AND	Rule 241	
113	Rule 180	Rule 191	AND	Rule 244	
114	Rule 181	Rule 191	AND	Rule 245	
115	Rule 182	Rule 178	OR	Rule 4	
116	Rule 183	Rule 191	AND	Rule 247	
117	Rule 184	Rule 186	AND	Rule 248	
118	Rule 185	Rule 187	AND	Rule 253	
119	Rule 188	Rule 191	AND	Rule 252	
120	Rule 189	Rule 191	AND	Rule 253	
121	Rule 190	Rule 191	AND	Rule 254	
122	Rule 193	Rule 205	AND	Rule 241	
123	Rule 194	Rule 206	AND	Rule 242	
124	Rule 195	Rule 207	AND	Rule 243	
125	Rule 197	Rule 205	AND	Rule 213	
126	Rule 198	Rule 206	AND	Rule 247	
127	Rule 199	Rule 207	AND	Rule 247	
128	Rule 201	Rule 205	AND	Rule 251	
129	Rule 202	Rule 206	AND	Rule 251	
130	Rule 203	Rule 207	AND	Rule 251	
131	Rule 209	Rule 213	AND	Rule 241	
132	Rule 210	Rule 223	AND	Rule 242	
133	Rule 211	Rule 223	AND	Rule 243	
134	Rule 214	Rule 212	OR	Rule 2	
135	Rule 215	Rule 223	AND	Rule 247	
136	Rule 216	Rule 220	AND	Rule 250	
137	Rule 217	Rule 221	AND	Rule 251	
138	Rule 218	Rule 223	AND	Rule 250	
139	Rule 219	Rule 223	AND	Rule 251	
140	Rule 222	Rule 223	AND	Rule 254	
141	Rule 225	Rule 239	AND	Rule 241	
142	Rule 226	Rule 239	AND	Rule 242	
143	Rule 227	Rule 239	AND	Rule 243	
144	Rule 228	Rule 239	AND	Rule 244	
145	Rule 229	Rule 239	AND	Rule 245	
146	Rule 230	Rule 238	AND	Rule 247	
147	Rule 231	Rule 239	AND	Rule 247	
148	Rule 233	Rule 232	OR	Rule 1	
149	Rule 235	Rule 239	AND	Rule 251	
150	Rule 237	Rule 239	AND	Rule 253	
151	Rule 246	Rule 247	AND	Rule 254	
152	Rule 249	Rule 251	AND	Rule 253	

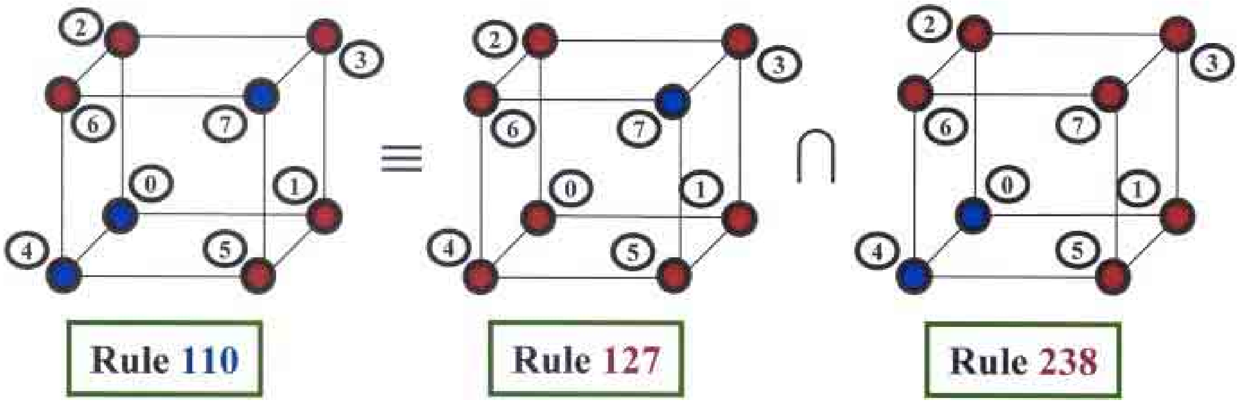


Fig. 22. Rule 110 = Rule 127 AND Rule 238.

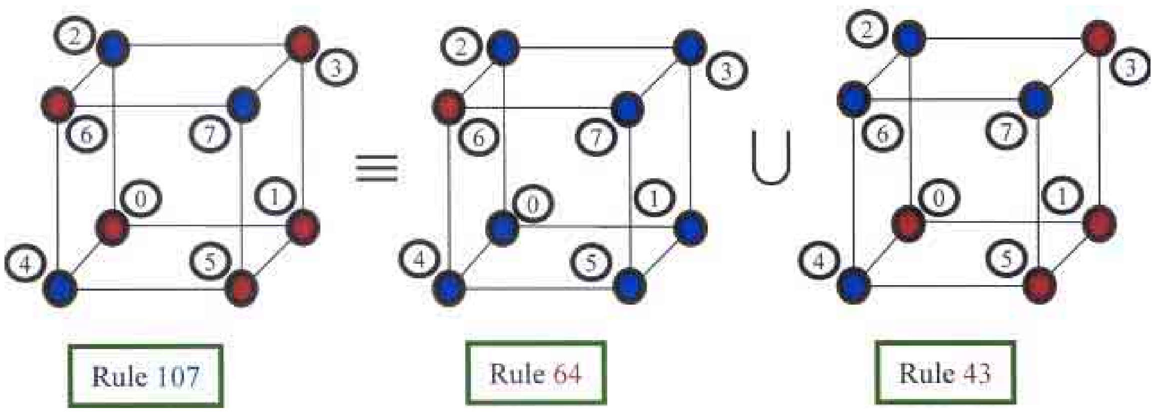


Fig. 23. Rule 107 = Rule 64 OR Rule 43.

Non-Separable local Rules. All together, it suffices to conduct an in-depth research on the nonlinear dynamics and "global" complexity of only 80 independent local Boolean Function of three binary variables.

5.8. *Making Non-Separable from Separable Rules*

Linearly Separable local Rules have a *complexity index* $\kappa = 1$, by definition. These are the simplest building blocks in the universe of Boolean cubes, of *any* dimension. They are also the simplest to implement on a *chip*.[15] In terms of their nonlinear dynamics, Linearly Separable Rules are also the *fastest* to execute on a chip; namely, a few nanoseconds via current silicon technology, and at the *speed of light* via current optical technology. Moreover, the speed of the associated Cellular Automata is independent of the size of the array — it takes *the same amount* of time to run a two-dimensional Linearly Separable Rule on a 10×10 array, or on a $10^6 \times 10^6$ array of Cellular Automata when executed on a CNN chip.

It is proved in [Chua & Roska, 2002] that *every* one or two-dimensional Linearly Non-Separable Boolean Rule can be implemented by combining only a *finite* number of Linearly Separable Rules via standard *logic operations* (AND, OR, and XOR) on each pixel of a CNN.[16] As the simplest special

[15] All 104 local Rules are implemented on the *CNN universal chip* directly on hardware, i.e. without programming [Chua & Roska, 2002].
[16] Every one of the $2^{2^9} = 2^{512} \approx 10^{154}$ distinct two-dimensional local Boolean Rules with nine inputs can be implemented directly on current *CNN universal chips* by programming via a C-like user-friendly language [Chua & Roska, 2002].

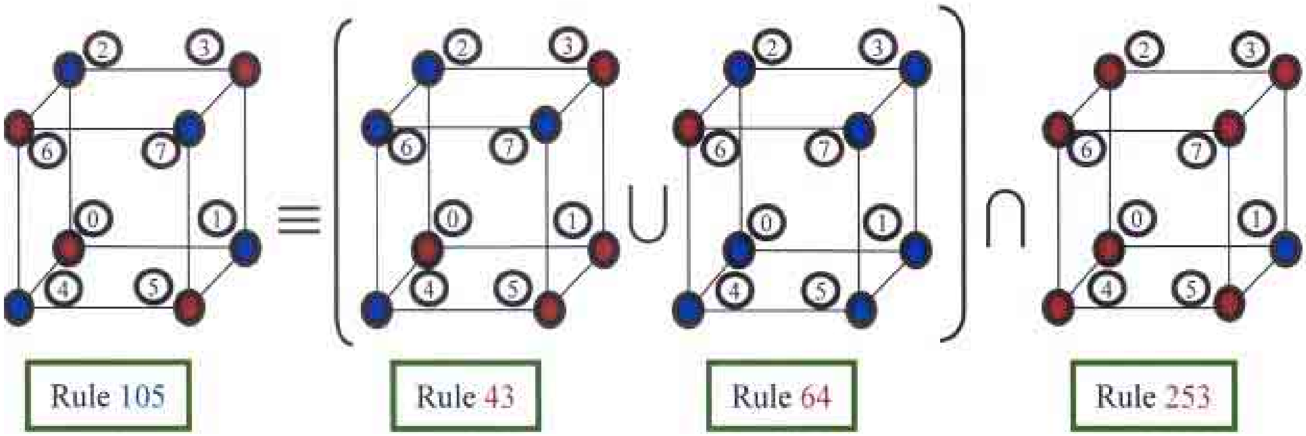


Fig. 24. Rule 105 = (Rule 43 OR Rule 64) AND Rule 253.

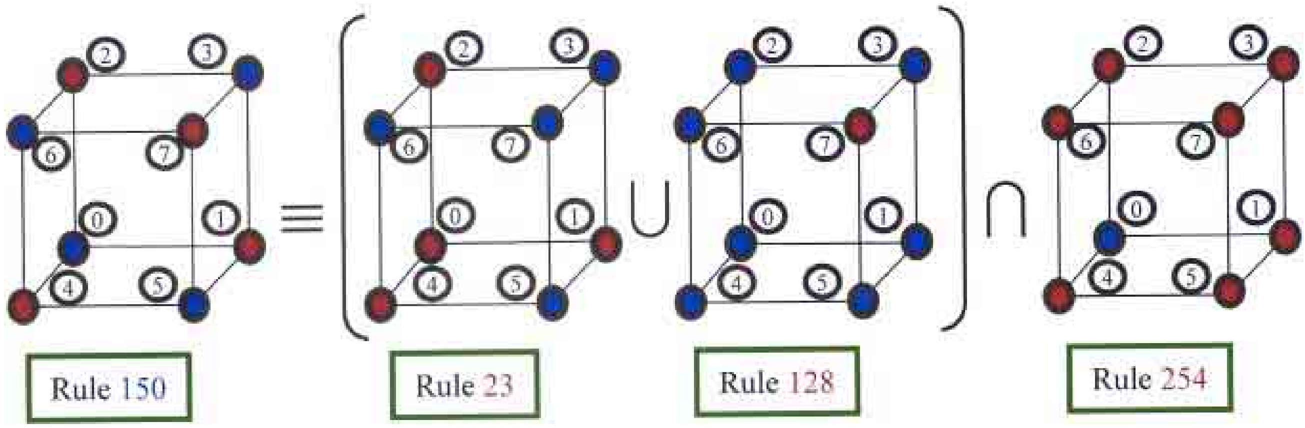


Fig. 25. Rule 150 = (Rule 23 OR Rule 128) AND Rule 254.

case of this fundamental *decomposition theorem*, Table 17 gives the explicit decomposition of all 152 Linearly Non-Separable local Rules from Table 10 in terms of at most three Linearly Separable Rules and combining them *pixelwise* only via AND and OR logic operations. An inspection of this table shows that in fact with the exception of Rule 105 and Rule 150, which require three Linearly Separable building blocks, all others need only two. In this sense, one could rank Rule 105 and Rule 150 as the most complicated one-dimensional Cellular Automata cells to implement on a chip [Dogaru & Chua, 1998].

The reader can easily verify the decompositions in Table 17 by performing the prescribed logic operation directly on corresponding vertices of the relevant Boolean cubes extracted from Table 1. Four examples of such decompositions are shown in Figs. 22–25 for Rule 110 (involving only one AND operation), Rule 107 (involving only one OR operation), and Rule 105 and *Rule 150* (both involving one AND and one OR operations).

5.9. *Index 2 is the Threshold of Complexity*

By inspection of the patterns generated in Table 2 for the 33 Linearly Separable Rules listed in Table 15, and by invoking, the "equivalence class" classification in Table 14, we find that *no local Rule with complexity index* $\kappa = 1$ is capable of generating complex patterns, even for *random* initial conditions. It is clear therefore that in order to exhibit *emergence* and *complex* phenomena, such as those presented in [Chua, 1998],[17] a local Rule must have a *minimum* complexity index of $\kappa = 2$. In other words, borrowing the name from Wolfram, we can assert that *complexity index 2 is the threshold of complexity for one-dimensional Cellular Automata.* This *analytically-based* assertion is certainly consistent with the following *empirically-based* observation extracted from pages 105–106 of [Wolfram, 2002]:

> **The examples in this chapter suggest that if the rules for a particular system are sufficiently simple, then the system will only ever exhibit purely repetitive behavior. If the rules are slightly more complicated, then nesting will also often appear. But to get complexity in the overall behavior of a system one needs to go beyond some threshold in the complexity of its underlying rules.**
>
> **The remarkable discovery that we have made, however, is that this threshold is typically extremely low. And indeed in the course of this chapter we have seen that in every single one of the general kinds of systems that we have discussed, it ultimately takes only very simple rules to produce behavior of great complexity.**

Acknowledgments

This work is supported in part by the Finnish Technology Development Centre (TEKES), the Academy of Finland, and the Nokia Foundation, as well as by ONR grants N00014-01-1-0741 and N000-14-98-1-0594. The third author is in part supported by a Volkswagen Stiftung grant (Germany) within the program "Cooperation with Scientists and Engineers from Central and Eastern Europe". Part of this work was performed at the Electronic Circuit Design Laboratory of Helsinki University of Technology.

References

Chua, L. O. [1969] *Introduction to Nonlinear Network Theory* (McGraw-Hill Book Company, NY).

Chua, L. O., Hasler, M., Moschytz, G. S. & Neirynck, J. [1995] "Autonomous cellular neural networks: A unified paradigm for pattern formation and active wave propagation," *IEEE Trans. Circuits Syst.* **42**(10), 559–577.

Chua, L. O. [1998] *CNN: A Paradigm for Complexity* (World Scientific, Singapore).

Chua, L. O. & Roska, T. [2002] *Cellular Neural Networks and Visual Computing: Foundations and Applications* (Cambridge University Press).

Dogaru, R. & Chua, L. O. [1998] "CNN genes for one-dimensional cellular automata: A multi-nested piecewise-linear approach," *Int. J. Bifurcation and Chaos* **8**(10), 1987–2001.

Dogaru, R. & Chua L. O. [1999] "Universal CNN cells," *Int. J. Bifurcation and Chaos* **9**, 1–48.

Liñan, G., Rodriguez-Vazquez, A., Espejo, S. & Dominguez-Castro, R. [2002] "ACE16K: A 128 × 128 focal plane analog processor with digital I/O," *Proc. 7th IEEE Int. Workshop on Cellular Neural Networks and their Applications*, ed. Tetzlaff, R. (World Scientific, Singapore), pp. 132–139.

Shilnikov, L., Shilnikov, A., Turaev, D. & Chua, L., [1998] *Methods of Qualitative Theory in Nonlinear Dynamics (Part I)* (World Scientific, Singapore).

Shilnikov, L., Shilnikov, A., Turaev, D. & Chua, L., [2001] *Methods of Qualitative Theory in Nonlinear Dynamics (Part II)* (World Scientific, Singapore).

Wolfram, S. [2002] *A New Kind of Science* (Wolfram Media, Inc., Champaign IL, USA).

[17] Virtually all of the complexity patterns from [Wolfram, 2002] could be generated by using only a standard CNN cell with 3×3 or 5×5 neighborhood size.