# 2

# A Comparative Study

As we pointed out in the previous chapter, several frameworks and constructions relate to SDS, and in the following we present a short overview. This chapter is not intended to be a complete survey — the list of frameworks that we present is not exhaustive, and for the concepts that we discuss we only provide enough of an introduction to allow for a comparison to SDS. Specifically, we discuss *cellular automata*, *random Boolean networks*, and *finite-state machines*. Other frameworks related to SDS that are not discussed here include *interacting particle systems* [25] and *Petri nets* [26].

## 2.1 Cellular Automata

### 2.1.1 Background

Cellular automata, or CA[1] for short, were introduced by von Neumann and Ulam around 1950 [27]. The motivation for CA was to obtain a better formal understanding of biological systems that are composed of many identical components and where each component is relatively simple, at least as compared to the full system. The design and structure of the first computers were another motivation for the introduction of CA.

The global dynamics or pattern evolution of a cellular automaton is the result of interactions of its components or cells. Questions such as to which patterns can occur for a given CA (computational universality) and which CA that, in an appropriate sense, can be used to construct descriptions of other CA (universal construction) were central in the early phases [27, 28]. Cellular automata have been studied from a dynamical systems perspective (see, for example, [29–33]), from a logic, automata, and language theoretic perspective (e.g., [28, 34, 35]), and through ergodic theory and in probabilistic settings

[1] Just as for SDS we use the abbreviation CA for both the singular and plural forms. It will be clear from the context which form is meant.

(e.g., [36–39]). Applications of cellular automata can be found, for example, in the study of biological systems (see [40]), in hydrodynamics in the form of lattice gases (see, for example, [41–43]), in information theory, and in the construction of codes [44], and in many other areas. For further details and overviews of the history and theory of CA, we refer to, e.g., [18, 45–47].

### 2.1.2 Structure of Cellular Automata

Cellular automata have many features in common with SDS. There is an underlying cell or lattice structure where each lattice point or cell $v$ has a state state $x_v$ taken from some finite set. Each lattice point has a function defined over a collection of states associated to nearby lattice points. As a dynamical system, a cellular automaton evolves in discrete time steps by the synchronous application of the cell functions.

Notice that the lattice structure is generally not the same as the base graph of SDS. As we will explain below, the notion of what constitutes adjacent vertices is determined by the lattice structure *and* the functions. Note that in contrast to SDS it is not uncommon to consider cellular automata over infinite lattices.

One of the central ideas in the development of CA was uniform structure, and in particular this includes translation invariance. As a consequence of this, the lattice is typically regular such as, for example, $\mathbb{Z}^k$ for $k \geq 1$. Moreover, translation invariance also implies that the functions $f_v$ and the state spaces $S_v$ are the same for all lattice points $v$. Thus, there are a common function $f$ and a common set $S$ such that $f_v = f$ and $S_v = S$ for all $v$. Additionally, the set $S$ usually has some designated zero element or *quiescent state* $s_0$. Note that in the study of CA dynamics over infinite structures like $\mathbb{Z}^k$, one considers the system states[2] $x = (x_v)_v$ where only a finite number of the cell states $x_v$ are different from $s_0$. Typically, $S = \{0, 1\}$ and $s_0 = 0$.

Each vertex $v$ in $Y$ has a *neighborhood* $n[v]$, which is some sequence of lattice points. Again for uniformity reasons all the neighborhoods $n[v]$ exhibit the same structure. In the case of $\mathbb{Z}^k$ the neighborhood is constructed from a sequence $N = (d_1, \ldots, d_m)$ where $d_i \in \mathbb{Z}^k$, and each neighborhood is given as $n[v] = v + N = (v + d_1, \ldots, v + d_m)$. A *global CA state*, system state, or CA configuration is an element $x \in S^{\mathbb{Z}^k}$. For convenience we write $x[v] = (x_{v+d_1}, \ldots, x_{v+d_m})$ for the subconfiguration associated with the neighborhood $n[v]$.

**Definition 2.1 (Cellular automata over $\mathbb{Z}^k$).** Let $S$, $N$, and $f$ be as above. The cellular automaton with states in $S$, neighborhood $N$, and function $f$ is the map

$$\Phi_f \colon S^{\mathbb{Z}^k} \longrightarrow S^{\mathbb{Z}^k}, \quad \Phi_f((x))_v = f(x[v]). \tag{2.1}$$

[2] For cellular automata a system state $x = (x_v)_v$ is usually called a configuration.

In other words, the cellular automaton dynamics results from the synchronous or parallel application of the maps $f$ to the cell states $x_v$.

We can also construct CA over finite lattices. One standard way to do this is by imposing *periodic boundary conditions*. In one-dimension we can achieve this by identifying vertices $i$ and $i + n$ in $\mathbb{Z}$ for some $n > 1$. This effectively creates a CA over $\mathbb{Z}/n\mathbb{Z}$. Naturally we can extend this to higher dimensions, in which case we would consider $k$-dimensional tori.

Another way to construct a CA over a finite structure is through *zero boundary conditions*. In one-dimension this means we would use the line graph $\mathsf{Line}_n$ as lattice and add two additional vertices at the ends and fix their states to zero; see Example 2.2.

*Example 2.2 (One-dimensional CA).* This example shows the three different types of graph or grid structures for one-dimensional CA that we discussed in the text. If we use the neighborhood structure given by $N = (-1, 0, 1)$, we see that to compute the new state for a cell $v$ the map $f$ only takes as arguments the state of the cell $v$ and the states of the nearest neighbors of $v$. For this reason this class of maps is often referred to as *nearest-neighbor rules*. The corresponding lattices are shown in Figure 2.1. ◇

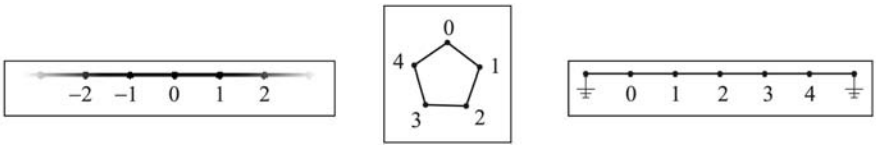


**Fig. 2.1.** From left to right: the lattice of a CA in the case of (a) $\mathbb{Z}$, (b) $\mathbb{Z}/5\mathbb{Z}$ with periodic boundary conditions, and (c) $\mathbb{Z}/5\mathbb{Z}$ with zero boundary conditions.

Two of the commonly used neighborhood structures $N$ are the *von Neumann neighborhood* and the *Moore neighborhood*. These are shown in Figure 2.2. For $\mathbb{Z}^2$ the von Neumann neighborhood is

$$N = ((0,0), (-1,0), (0,-1), (1,0), (0,1)) \,.$$

The *radius* of a one-dimensional CA rule $f$ with neighborhood defined by $N$ is the norm of the largest element of $N$. The radius of the rule in Example 2.2 is therefore 1.

We see that the lattice and the function of a cellular automaton give us an $\mathsf{SDS}$ base graph $Y$ as follows. For the vertices of $Y$ we take all the cells. A vertex $v$ is adjacent to all vertices $v'$ in $n[v]$. If $v$ itself is included in $n[v]$, we make the convention of omitting the loop $\{v, v\}$.

In analogy to $\mathsf{SDS}$, one central goal of CA research is to derive as much information as possible about the global dynamics of the CA map $\Phi_f$ based on known, local properties such as the map $f$ and the neighborhood structure.

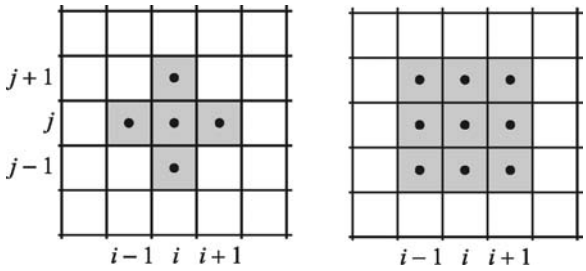


**Fig. 2.2.** The von Neumann neighborhood (left) and the Moore neighborhood (right) for an infinite two-dimensional CA.

The *phase space* of a CA is the directed graph with all possible configurations as vertices, and where vertices $x$ and $y$ are connected by a directed edge $(x, y)$ if $\Phi_f(x) = y$. Even in the case of CA over finite lattices, it is impractical to display the whole phase space, and *space-time diagrams* (see Section 4.1) are often used to visualize certain orbits or trajectories.

*Example 2.3.* The CA rule $f_{90}$ is given by $f_{90}(x_{i-1}, x_i, x_{i+1}) = x_{i-1} + x_{i+1}$ modulo 2. This *linear* function has been studied extensively in, for example, [32]. In Figure 2.3 we have shown two typical space-time diagrams for the CA with local rule $f_{90}$ over the lattice $\mathsf{Circ}_{512}$. ◇

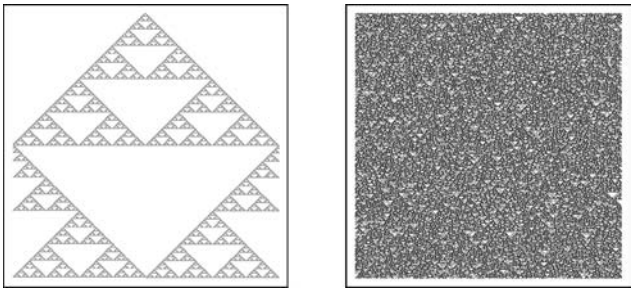

**Fig. 2.3.** Space-time diagrams for CA with cell function $f_{90}$. In the left diagram the initial configuration contains a single state that is 1. In the right diagram the initial configuration was chosen at random.

CA differ from SDS in several ways. For instance, for CA the graph $Y$, which is derived from the lattice and neighborhood $n[v]$, is regular and translation invariant, whereas the graph of an SDS is arbitrary, although finite. Furthermore, CA have a fixed function or rule, associated to every vertex, while SDS have a vertex-indexed family of functions. Perhaps most importantly, CA and SDS differ in their respective update schemes. As a result, CA and SDS differ significantly with respect to, for example, invertibility as we will show in the exercises.

In principle one can generalize the concept of CA and consider them over arbitrary graphs with vertex-indexed functions. One may also consider asynchronous CA. The dynamics of the latter class of CA depends critically on the particular choice of update order [48].

In the remainder of this section we will give a brief account of some basic facts and terminology on CA that will be used in the context of SDS.

### 2.1.3 Elementary CA Rules

A large part of the research on CA has been concerned with the finite and infinite one-dimensional cases where the lattice is $\mathbb{Z}/n\mathbb{Z}$ and $\mathbb{Z}$, respectively. An example of a phase space of a one-dimensional CA with periodic boundary conditions is shown in Figure 2.1. The typical setting uses radius-1 vertex functions with binary states. In other words, the functions are of the form $f\colon \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2$ where $\mathbb{F}_2 = \{0, 1\}$ is the field with two elements. Whether the lattice is $\mathbb{Z}$ or $\mathbb{Z}/n\mathbb{Z}$, we refer to this class of functions as the *elementary CA rules* and the corresponding global CA maps as elementary CA.

*Example 2.4.* Let $\Phi_f$ be the CA with local rule $f\colon \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2$ given by $f(x, y, z) = (1 + y)(1 + z) + (1 + xyz)$. In this case we see that the state $(1, 0, 1, 1)$ maps to $(1, 1, 1, 0)$. The phase space of $\Phi_f$ is shown in Figure 2.4. ◇

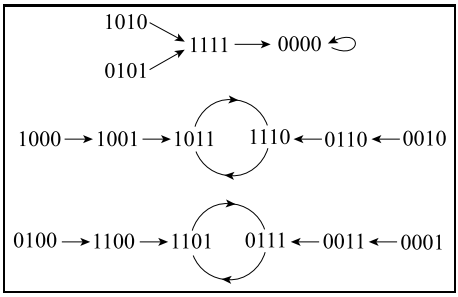


**Fig. 2.4.** The phase space of the elementary CA of Example 2.4.

### Enumeration of Elementary CA Rules

Clearly, there are $|\mathbb{F}_2|^{|\mathbb{F}_2^3|} = 2^8 = 256$ elementary CA rules. Any such function or rule $f$ can be specified as in Table 2.1 by the values $a_0$ through $a_7$. We identify the triple $x = (x_2, x_1, x_0) \in \mathbb{F}_2^3$ with the decimal number $k = k(x) = x_2 \cdot 2^2 + x_1 \cdot 2 + x_0$. Let the value of $f$ at $x$ be $a_k$ for $0 \leq k \leq 7$.[3] We can then encode the map $f$ as the decimal number $r = r(f)$ with $0 \leq r \leq 255$ through

[3] In the literature the $a_i$'s are sometimes ordered the opposite way.

| $(x_{i-1}, x_i, x_{i+1})$ | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|---|
| $f$ | $a_7$ | $a_6$ | $a_5$ | $a_4$ | $a_3$ | $a_2$ | $a_1$ | $a_0$ |

**Table 2.1.** Specification of elementary CA rules.

$$r = r(f) = \sum_{i=0}^{7} a_i 2^i \ . \tag{2.2}$$

This assignment of a decimal number in $\{0, 1, 2, \ldots, 255\}$ to the rule $f$ was popularized by S. Wolfram, and it is often referred to as the *Wolfram enumeration* of elementary CA rules [47, 49]. This enumeration procedure can be generalized to other classes of rules, and some of these are outlined in Problem 2.2.

*Example 2.5.* The map $\text{parity}_3 \colon \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2$ given by $\text{parity}_3(x_1, x_2, x_3) = x_1 + x_2 + x_3$ with addition modulo 2 (i.e., in the field $\mathbb{F}_2$) can be represented by

| $(x_{i-1}x_ix_{i+1})$ | 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 |
|---|---|---|---|---|---|---|---|---|
| parity | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |

and thus

$$r(\text{parity}_3) = 2^7 + 2^4 + 2^2 + 2 = 150 \ . \qquad \diamond$$

A lot of work has gone into the study of this rule [32], and it is often referred to as the XOR function or the parity function. One of the reasons this rule has attracted much attention is that the induced CA is a *linear CA*. As a result all the machinery from algebra and matrices over finite fields can be put to work [33, 50].

**2.1.** What is the rule number of the elementary CA rule in Example 2.4? **[1]**

### Equivalence of Elementary CA Rules

Clearly, all the elementary CA are different as functions: For different elementary rules $f_1$ and $f_2$ we can always find a system state $x$ such that the induced CA maps differ for $x$. However, as far as dynamics is concerned, many of the elementary rules induce cellular automaton maps where the phase spaces look identical modulo labels (states) on the vertices. The precise meaning of "look identical" is that their phase spaces are isomorphic, directed graphs as in Section 4.3.3. When the phase spaces are isomorphic, we refer to the corresponding CA maps as *dynamically equivalent*. Two cellular automata $\Phi_f$ and $\Phi_{f'}$ with states in $\mathbb{F}_2$ are dynamically equivalent if there exists a bijection $h \colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ such that

$$\Phi_{f'} \circ h = h \circ \Phi_f \ . \tag{2.3}$$

The map $h$ is thus a one-to-one correspondence of trajectories of $\Phi_f$ and $\Phi_{f'}$. Alternatively, we may view $h$ as a relabeling of the states in the phase space.

*Example 2.6.* The phase spaces of the elementary CA with local rules 124 and 193 are shown in Figure 2.5. It is easy to check that the phase spaces are isomorphic. Moreover, the phase spaces are also isomorphic to the phase space shown in Figure 2.4 for the elementary CA 110. ◇

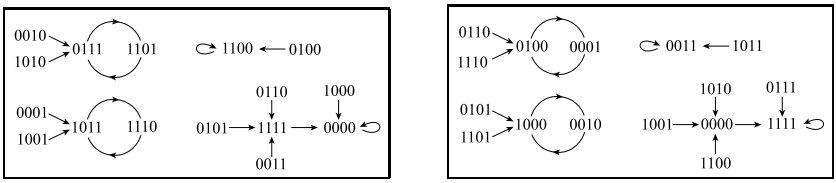


**Fig. 2.5.** The phase spaces of the elementary CA 124 (left) and 193 (right).

We will next show two things: (1) there at most 88 dynamically non-equivalent elementary CA, and (2) if we use a *fixed* sequential permutation update order rather than a synchronous update, then the corresponding bound for the number of dynamically non-equivalent systems is 136.

For this purpose we represent each elementary rule $f$ by a binary 8-tuple $(a_7, \ldots, a_0)$ (see Table 2.1) and consider the set

$$R = \{(a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0) \in \mathbb{F}_2^8\} \; . \tag{2.4}$$

Rules that give dynamically equivalent CA are related by two types of symmetries: (1) 0/1-flip symmetries (inversion) and (2) left-right symmetries. Let $\gamma \colon R \longrightarrow R$ be the map given by

$$\gamma(r = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)) = (\bar{a}_0, \bar{a}_1, \bar{a}_2, \bar{a}_3, \bar{a}_4, \bar{a}_5, \bar{a}_6, \bar{a}_7), \tag{2.5}$$

where $\bar{a}$ equals $1 + a$ computed in $\mathbb{F}_2$. With the map $\mathsf{inv}_n$ defined by

$$\mathsf{inv}_n \colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n, \quad \mathsf{inv}_n(x_1, \ldots, x_n) = (\bar{x}_1, \ldots, \bar{x}_n) \tag{2.6}$$

(note that $\mathsf{inv}_n^2 = \mathsf{id}$), a direct calculation shows that

$$\Phi_{\gamma(f)} = \mathsf{inv} \circ \Phi_f \circ \mathsf{inv}^{-1} \; ;$$

hence, 0/1-flip symmetry yields isomorphic phase spaces for $\Phi_f$ and $\Phi_{\gamma(f)}$.

As for left-right symmetry, we introduce the map $\delta \colon R \longrightarrow R$ given by

$$\delta(r = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)) = (a_7, a_3, a_5, a_1, a_6, a_2, a_4, a_0) \; . \tag{2.7}$$

The $\mathsf{Circ}_n$-automorphism $i \mapsto n + 1 - i$ induces in a natural way the map

$$\mathsf{rev}_n \colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n, \quad \mathsf{rev}_n(x_1, \ldots, x_n) = (x_n, \ldots, x_1) \tag{2.8}$$

(note $\mathsf{rev}_n^2 = \mathsf{id}$), and we have

$$\Phi_{\delta(f)} = \mathsf{rev} \circ \Phi_f \circ \mathsf{rev}^{-1} \; .$$

*Example 2.7 (Left-right symmetry).* The map defined by $f(x_1, x_2, x_3) = x_3$ induces a CA that acts as a left-shift (or counterclockwise shift if periodic boundary conditions are used). It is the rule $r = (1, 0, 1, 0, 1, 0, 1, 0)$ and it has Wolfram encoding 170. For this rule we have $\delta(r) = (1, 1, 1, 1, 0, 0, 0, 0)$, which is rule 240. We recognize this rule as the map $f(x_1, x_2, x_3) = x_1$, which is the rule that induces the "right-shift CA" as you probably expected. ◇

In order to compute the number of non-equivalent elementary CA, we consider the group $G = \langle \gamma, \delta \rangle$. Since $\gamma \circ \delta = \delta \circ \gamma$ and $\delta^2 = \gamma^2 = 1$, we have $G = \{1, \gamma, \delta, \gamma \circ \delta\}$ and $G$ acts on $R$. The number of non-equivalent rules is bounded above by the number of orbits in $R$ under the action of $G$ and there are 88 such orbits.

**Proposition 2.8.** *For $n \geq 3$ there are at most* 88 *non-equivalent phase spaces for elementary cellular automata.*

*Proof.* By the discussion above the number of orbits in $R$ under the action of $G$ is an upper bound for the number of non-equivalent CA phase spaces. By the Frobenius lemma [see (3.18)], this number is given by

$$N = \frac{1}{4} \sum_{\eta \in G} |\mathsf{Fix}(g)| = \frac{1}{4}(|\mathsf{Fix}(1)| + |\mathsf{Fix}(\gamma)| + |\mathsf{Fix}(\delta)| + |\mathsf{Fix}(\gamma \circ \delta)|) \; . \quad (2.9)$$

We leave the remaining computations to the reader as Problem 2.2. □

**2.2.** Compute the terms $|\mathsf{Fix}(1)|$, $|\mathsf{Fix}(\gamma)|$, $|\mathsf{Fix}(\delta)|$, and $|\mathsf{Fix}(\gamma \circ \delta)|$ in (2.9) and verify that you get $N = 88$. **[1]**

Note that we have not shown that the bound 88 is a sharp bound. That is another exercise — it may take some patience.

**2.3.** Is the bound 88 for the number of dynamically non-equivalent elementary CA sharp? That is, if $f$ and $g$ are representative rules for different orbits in $R$ under $G$, then are the phase spaces of $\Phi_f$ and $\Phi_g$ non-isomorphic as directed graphs? **[3]**

*Example 2.9.* Consider the elementary CA rule numbered 14 and represented as $r = (0, 0, 0, 0, 1, 1, 1, 0)$. In this case we have $G(r) = \{r, \gamma(r), \delta(r), \gamma \circ \delta(r)\} = \{r_{14}, r_{143}, r_{84}, r_{214}\}$ using the Wolfram encoding. ◇

**2.4.** (a) What is $R^G$ (the set of elements in $R$ fixed by all $g \in G$) for the action of $G$ on the elementary CA rules $R$ in (2.4)?
(b) Do left-right symmetric elementary rules induce equivalent permutation-SDS? That is, for a fixed sequential permutation update order $\pi$, do we get equivalent global update maps? What happens if we drop the requirement of a fixed permanent updates order?
(c) What is the corresponding transformation group $G'$ acting on elementary rules in the case of SDS with a fixed update order $\pi$? How many orbits are there in this case?
(d) Show that $R^G = R^{G'}$. **[2-C]**

**Other Classes of CA Rules**

In addition to elementary CA rules, the following particular classes of CA rules are studied in the literature: the *symmetric rules*, the *totalistic rules*, and the radius-2 rules. Recall that a function $f\colon K^n \longrightarrow K$ is symmetric if for every permutation $\sigma \in S_n$ we have $f(\sigma \cdot x) = f(x)$ where $\sigma \cdot (x_1, \ldots, x_n) = (x_{\sigma^{-1}(1)}, \ldots, x_{\sigma^{-1}(n)})$. Thus, a symmetric rule $f$ does not depend on the order of its argument. A totalistic function is a function that only depends on $(x_1, \ldots, x_n)$ through the sum $\sum x_i$ (taken in $\mathbb{N}$). Of course, over $\mathbb{F}_2$ symmetric and totalistic rules coincide. The radius-2 rules are the rules of the form $f\colon K^5 \longrightarrow K$ that are used to map $(x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2})$ to the new state $x_i'$ of cell $i$.

In some cases it may be natural or required that we handle the state of a vertex $v$ differently than the states of its neighbor vertices when we update the state $x_v$. If the map $f$ used to update the state $v$ is symmetric in the arguments corresponding to the neighbor states of cell $v$, we call $f_v$ *outer-symmetric*.

The classes of *linear CA over finite fields* and general linear maps over finite fields have been analyzed extensively in, e.g., [32,33,50,51]. Let $K$ be a field. A map $f\colon K^n \longrightarrow K$ is linear if for all $\alpha, \beta \in K$ and all $x, y \in K^n$ we have $f(\alpha x + \beta y) = \alpha f(x) + \beta f(Y)$. A CA induced by a linear rule is itself a linear map. Linear maps over rings have been studied in [52].

*Example 2.10.* The elementary CA rule 90, which is given as $f_{90}(x_1, x_2, x_3) = x_1 + x_3$, is outer-symmetric but not totalistic or symmetric. The elementary CA rule $g(x_1, x_2, x_3) = (1 + x_1)(1 + x_2)(1 + x_3)$, which is rule 1, is totalistic and symmetric. Note that the first rule is linear, whereas the second rule is nonlinear. ◇

*Example 2.11.* A space-time diagram of a radius-2 rule is shown in Figure 2.6. By using the straightforward extension of Wolfram's encoding to this class of CA rules, we see that this particular rule has encoding 3283936144, or $(195, 188, 227, 144)$ in the notation of [53]. ◇

In the case of linear CA over $\mathbb{Z}/n\mathbb{Z}$, we can represent the CA map through a matrix $A \in K^{n \times n}$. This means we can apply algebra and finite field theory to analyze the corresponding phase spaces through normal forms of $A$. We will not go into details about this here — a nice overview can be found in [33]. We content ourselves with the following result.

**Theorem 2.12 ( [33]).** *Let $K$ be a finite field of order $q$ and let $M \in K^{n \times n}$. If the dimension of $\ker(M)$ is $k$, then there is a rooted tree $T$ of size $q^k$ such that the phase space of the dynamical system given by the map $F(x) = Mx$ consists of $q^{n-k}$ cycle states, each of which has an isomorphic copy of $T$ attached at the root vertex.*

In other words, for a finite linear dynamical system over a field, all the transient structures are identical.

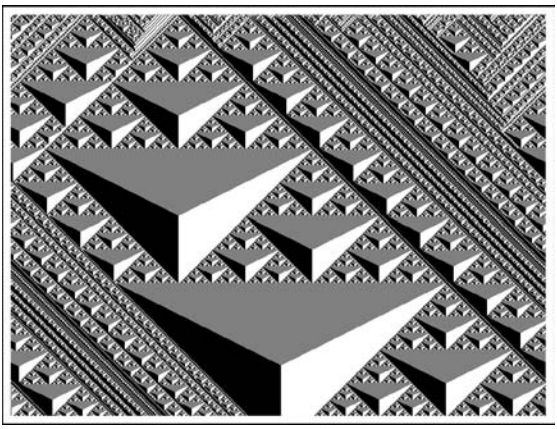

**Fig. 2.6.** A space-time diagram for the radius-2 CA over $\mathbb{Z}/1024\mathbb{Z}$ with rule number 3283936144 starting from a randomly chosen initial state.

**2.5.** Consider the finite linear dynamical system $f\colon \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$ with matrix (relative to standard basis)

$$M = \begin{bmatrix} 0\,1\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,1\,1 \\ 0\,0\,1\,0 \end{bmatrix} .$$

Show that the phase space consists of one fixed point and one cycle of length three. Also show that the transient tree structures at the periodic points are all identical. **[1]**

**2.6.** Use the elementary CA 150 over $\mathbb{Z}/n\mathbb{Z}$ to show that the question of whether or not a CA map is invertible depends on $n$. (As we will see in Chapter 4, this does not happen with a sequential update order.) **[1+C]**

**2.7.** How many linear, one-dimensional, elementary CA rules of radius $r$ are there? Give their Wolfram encoding in the case $r = 1$. **[1+]**

**2.8.** How many elementary CA rules $f\colon \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2$ satisfy the symmetry condition

$$f(x_{i-1}, x_i, x_{i+1}) = f(x_{i+1}, x_i, x_{i-1})$$

and the quiescence condition

$$f(0, 0, 0) = 0\ ?$$

An analysis of the cellular automata induced by these rules can be found in, e.g., [32, 49]. **[1]**

## 2.2 Random Boolean Networks

*Boolean networks* (BN) were originally introduced by S. Kauffman [54] as a modeling framework for gene-regulatory networks. Since their introduction some modifications have been made, and here we present the basic setup as given in, e.g., [55–58], but see also [59].

A Boolean network has vertices or genes $V = \{v_1, \ldots, v_n\}$ and functions $F = (f_1, \ldots, f_n)$. Each gene $v_i$ is linked or "wired" to $k_i$ genes as specified by a map $e_i \colon \{1, \ldots, k_i\} \longrightarrow V$. The Boolean state $x_{v_i}$ of each gene is updated as

$$x_{v_i} \mapsto f_i(x_{e_i(1)}, \ldots, x_{e_i(k_i)}) \ ,$$

and the whole state configuration is updated synchronously. Traditionally, the value of $k_i$ was the same for all the vertices. A gene or vertex $v$ that has state 1 is said to be expressed.

*A random Boolean network* (RBN) can be obtained in the following ways. First, each vertex $v_i$ is assigned a sequence of maps $f^i = (f^i_1, \ldots, f^i_{l_i})$. At each point $t$ in time a function $f^i_t$ is chosen from this sequence for each vertex at random according to some distribution. The *function configuration* $(f^1_t, \ldots, f^n_t)$ that results is then used to compute the system configuration at time $t+1$ based on the system configuration at time $t$. Second, we may consider for a fixed function $f_i$ over $k_i$-variables the map $e_i \colon \{1, \ldots, k_i\} \longrightarrow V$ to be randomly chosen. That amounts to choosing a random directed graph in which $v_i$ has in-degree $k_i$.

Since random Boolean networks are stochastic systems, they cannot be described using the traditional phase-space notion. As you may have expected, the framework of *Markov chains* is a natural way to capture their behavior. The idea behind this approach is straightforward and can be illustrated as follows.

Let $0 \leq p \leq 1.0$ and let $i \in \mathbb{Z}/n\mathbb{Z}$ be a vertex of an elementary CA (see the previous section) with update function $f$ and states in $\{0, 1\}$. Let $f'$ be some other elementary CA function. If we update vertex $i$ using the function $f$ with probability $p$ and with function $f'$ with probability $(1-p)$ and use the function $f$ for all other vertices states, we have a very basic random Boolean network. This stochastic system may be viewed as a weighted superposition of two deterministic cellular automata. By this we mean the following: If the state of vertex $i$ is always updated using the map $f$, we obtain a phase space $\Gamma$, and if we always update the state of vertex $i$ using the function $f'$, we get a phase space $\tilde{\Gamma}$. The weighted sum "$p\Gamma + (1-p)\tilde{\Gamma}$" is the directed, weighted graph with vertices all states of state space, with a directed edge from $x$ to $y$ if any of the two phase spaces contains this transition. The weight of the edge $(x, y)$ is $p$ (respective, $1-p$) if only $\Gamma$ (respective, $\tilde{\Gamma}$) contains this transition, and 1 if both phase spaces contain the transition. In general, the weight of the edge $(x, y)$ is the sum of the probabilities of the configurations that has an associated phase space, which includes this transition. We may

call the resulting weighted graph the *probabilistic phase space*. The evolution of the random Boolean network may therefore be viewed as a random walk on the probabilistic phase space. The corresponding weighted adjacency matrix directly and naturally encodes the associated Markov chain matrix of the RBN.

This Markov chain approach is the basis used for the framework of random Boolean networks as studied by, e.g., Shmulevich and Dougherty [55]. The following example provides a specific illustration.

*Example 2.13.* Let $Y = \mathsf{Circ}_3$ and, with the exception of $f_0$, let each function $f_i$ be induced by $\mathrm{nor}_3 \colon \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2$. For $f_0$ we use $\mathrm{nor}_3$ with probability $p = 0.4$ and $\mathrm{parity}_3$ with probability $q = 1 - p$. In the notation above we get the phase spaces $\Gamma$, $\tilde{\Gamma}$, and $p\Gamma + (1-p)\tilde{\Gamma}$ as shown in Figure 2.7. ◇

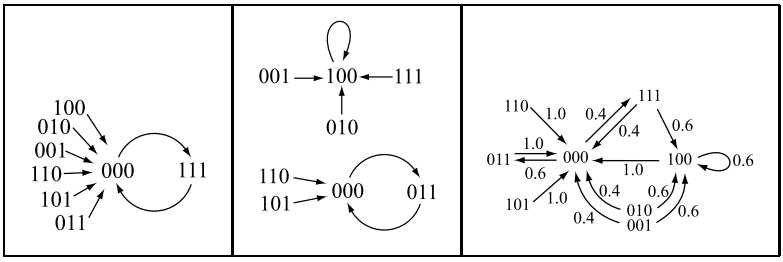


**Fig. 2.7.** The phase spaces $\Gamma$, $\tilde{\Gamma}$, and $p\Gamma + (1-p)\tilde{\Gamma}$ of Example 2.13.

The concept of Boolean networks resembles several features of SDS. For instance, an analogue of the SDS dependency graph can be derived via the maps $e_i$. However, research on Boolean networks focuses on analyzing the functions, while for SDS the study of graph properties and update orders is of equal importance. As for sequential update schemes, we remark that aspects of asynchronous RBN have been studied in [60].

## 2.3 Finite-State Machines (FSMs)

*Finite-state machines* (FSM) [61–63] and their extensions constitute another theory and application framework. Their use ranges from tracking and response of weapon systems to dishwasher logic and all the way to the "AI-logic" of "bots" or "enemies" in computer games. Finite-state machines are not dynamical systems, but they do exhibit similarities with both SDS and cellular automata.

**Definition 2.14.** A finite-state machine (or a finite automaton) is a five-tuple $M = (K, \Sigma, \tau, x_0, A)$ where $K$ is a finite set (the *states*), $\Sigma$ is a finite set (the *alphabet*), $\tau \colon K \times \Sigma \longrightarrow K$ is the *transition function*, $x_0 \in K$ is the *start state*, and $A \subset K$ is the *set of accept states*.

Thus, for each state $x \in K$ and for each letter $s \in \Sigma$ there is a directed edge $(x, x_s)$. The finite-state machine reads input from, e.g., an *input tape*. If the finite-state machine is in state $x$ and reads the input symbol $s \in \Sigma$, it will transition to state $x_s$. If at the end of the input tape the current state is one of the states from $A$, the machine is said to accept the input tape. One therefore speaks about the set of input tapes or sequences accepted by the machine. This set of accepted input sequences is the *language accepted by* $M$. An FSM is often represented pictorially by its *transition diagram*, which has the states as vertices and has directed edges $(x, \tau(x, s))$ labeled by $s$.

If the reading of a symbol and the subsequent state transition take place every time unit, we see that each input sequence $\sigma$ generates a time series of states $(M_\sigma(x_0, t))_{t=0}$. Here $M_\sigma(x_0, t)$ denotes the state at time $t$ under the time evolution of $M$ given the input sequence $\sigma$. The resemblance to finite dynamical systems is evident.

*Example 2.15.* In real applications the symbol may come in the form of events from some input system. A familiar example is traffic lights at a road intersection. The states in this case could be all permissible red–yellow–green configurations. A combination of a clock and vehicle sensors can provide events that are encoded as input symbols every second, say. The transition function implements the traffic logic, hopefully in a somewhat fair way and in accord with traffic rules. ◇

Our notion of all finite-state machine is often called a *deterministic finite-state machine* (DFSM), see, e.g., [61], where one can find in particular the equivalence of *regular languages* and finite-state machines.

## Problems

**2.9. Enumeration of CA rules**
How many symmetric CA rules of radius 2 are there for binary states? How many outer-totalistic CA rules of radius 2 are there over $\mathbb{F}_2$? How many outer-symmetric CA rules of radius $r$ are there with states in $\mathbb{F}_p$, the finite field with $p$ elements ($p$ prime)? **[1+]**

**2.10.** A *soliton* is, roughly speaking, a solitary localized wave that propagates without change in shape or speed even upon collisions with other solitary waves. Examples of solitons occur as solutions to several partial differential equations. In [64] it is demonstrated that somewhat similar behavior occur in *filter automata*.

The state space is $\{0, 1\}^{\mathbb{Z}}$. Let $x^t$ denote the state at time $t$. For a filtered automaton with radius $r$ and rule $f$ the successor configuration to $x^t$ is computed in a left-to-right (sequential) fashion as

$$x_i^{t+1} = f(x_{i-r}^{t+1}, \ldots, x_{i-1}^{t+1}, x_i^t, x_{i+1}^t, \ldots, x_{i+r}^t).$$

Argue, at least in the case of periodic boundary conditions, that a filter automaton is a particular instance of a sequential dynamical system.

Implement this system as a computer program and study orbits starting from initial states that contain a small number of states that are 1. Use the radius-3 and radius-5 functions $f_3$ and $f_5$ where $f_k \colon \mathbb{F}_2^{2k+1} \longrightarrow \mathbb{F}_2$ is given by

$$f_k(x_{-k}, \ldots, x_{-1}, x_0, x_1, \ldots, x_k) = \begin{cases} 0 & \text{if each } x_i \text{ is zero,} \\ \sum_{i=-k}^{k} x_i & \text{otherwise,} \end{cases}$$

where the summation is in $\mathbb{F}_2$. Note that these filter automata can be simulated by a CA; see [64]. **[1+C]**

## Answers to Problems

**2.1.** 110.

**2.2.** Every rule $(a_7, \ldots, a_0)$ is fixed under the identity element, so $|\mathsf{Fix}(1)| = 256$. For a rule to be fixed under $\gamma$ it must satisfy $(a_7, \ldots, a_0) = (\bar{a}_0, \ldots, \bar{a}_7)$, and there are $2^4$ such rules. Likewise there are $2^6$ rules fixed under $\delta$ and $2^4$ rules fixed under $\gamma \circ \delta$.

**2.4.** (b) No. The SDS of the left-right rule is equivalent to the SDS of the original rule but with a different update order. What is the update order relation? (c) $G' = \{1, \gamma\}$. There are 136 orbits.

**2.6.** Derive the matrix representation of the CA and compute its determinant (in $\mathbb{F}_2$) for $n = 3$ and $n = 4$.

**2.7.** $2^{2r+1}$.

**2.8.** $2^5$.

**2.9.** $(i)$ $2^6 = 64$. $(ii)$ $2^5 \cdot 2^5 = 2^{10} = 1024$. $(iii)$ $(2^{2r+1})^p$.

**2.10.** Some examples of orbits are shown in Figure 2.8.

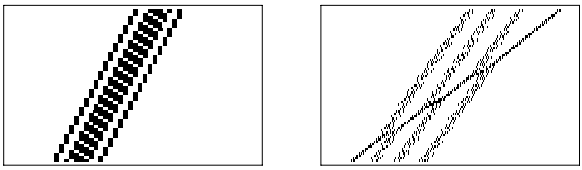

**Fig. 2.8.** "Solitions" in an automata setting. In the left diagram the rule $f_3$ is used, while in the right diagram the rule $f_5$ is used.

http://www.springer.com/978-0-387-30654-4

An Introduction to Sequential Dynamical Systems

Mortveit, H.; Reidys, C.

2008, XII, 248 p. 73 illus., Softcover
ISBN: 978-0-387-30654-4