

DIPLOMARBEIT / DIPLOMA THESIS

Titel der Diplomarbeit / Title of the Diploma Thesis

„Miracles of Simplicity – Cellular Automata and Selected Applications“

verfasst von / submitted by

Mag. Matthias Danzinger

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Magister der Naturwissenschaften (Mag. rer. nat.)

Wien, 2017/ Vienna, 2017

Studienkennzahl / Degree program code:

A 190 406 299

Studienrichtung / Degree program:

Lehramtsstudium UF Mathematik UF
Psychologie und Philosophie

Betreut von / Supervisor:

ao. Univ.-Prof. Mag. Dr. Peter Raith

Acknowledgements

I would like to thank Prof. Peter Raith not only for supervising this thesis and providing helpful advice but also for everything I could learn from you during my studies in mathematics. I am indebted to Katharina and Georg for proofreading.

Furthermore, I would like to thank my family. No matter what may come, I will always try my best to support you in the same way you supported me.

I would like to thank Katharina for being such a wonderful person. Il y a des fleurs partout pour qui veut bien les voir. Grâce à toi je les vois. Ensemble on les cherche.

I am thankful to all of my friends and colleagues. Hanae, Georg and Manuel, studying mathematics would not have been even half as fun without you.

Last but not least I would like to thank all of my teachers, especially my piano, composition and harmony teacher for offering me a glance into this beautiful world with so much patience.

*Danke euch allen!
Katharina*

Contents

Acknowledgements	iii
1 Introduction	1
1.1 Definition	3
1.1.1 Topology	3
1.1.2 Neighbourhood	4
1.1.3 State Set or Alphabet	7
1.1.4 Transition Functions	7
2 Basic Properties of Cellular Automata	9
2.1 Global Transition Functions, Orbits, Patterns and Orphans	9
2.2 Sequences and Convergence of Configurations	11
2.3 Quiescent State, S-Support and S-Finite Configurations	12
2.4 Basic Properties of Transition Functions	13
2.5 Moore-Myhill Theorem	16
2.6 Linear Cellular Automata	19
2.7 Basic Concepts of Complexity Theory	20
3 One Dimensional Cellular Automata	21
3.1 Rule Number as Description for the Local Function	21
3.2 Examples	22
3.3 Fractal Structures	23
3.3.1 Random Initial Conditions	27
3.4 Rule Classification Scheme	28
3.4.1 Elementary Class 1 Automata	29
3.4.2 Elementary Class 2 Automata	30

3.4.3	Elementary Class 3 Automata	30
3.4.4	Elementary Class 4 Automata	31
3.5	Further Discussion of the 4 Classes	31
3.5.1	Formalized Approach by CULIK and YU	31
3.6	Sequence of Numbers in Elementary Cellular Automata	32
3.6.1	The Jacobsthal Numbers	32
3.6.2	The Mersenne Numbers	34
3.7	Computational Universality in Elementary Cellular Automata	36
3.7.1	Turing Machines	36
3.7.2	Computational Universality of Rule 110	37
3.8	Automata with Extended Neighbourhood and State Set	42
3.8.1	Class 4 Rule with Extended Neighbourhood Radius	43
3.8.2	Finite Width Example and Computational Tests	46
4	Two Dimensional Cellular Automata	53
4.1	Overview	54
4.2	Game of Life	55
4.2.1	Mathematical Properties of the Transition Function	59
4.3	Langton's Vant	61
4.3.1	Theoretical Properties of Vant	64
4.4	Linear Rules	66
5	Selected Applications of Cellular Automata	71
5.1	Applications of Cellular Automata in Cryptography	71
5.1.1	Random Number Generation	71
5.1.2	An Asymmetric Encryption Mechanism using Reversible Cellular Automata	74
5.2	Simple Applications of Linear Rules in Image Processing	75
5.2.1	Translation	75
5.2.2	Other Basic Image Manipulations and Sweeper's Algorithm	76
5.3	Edge Detection with Cellular Automata	77
5.3.1	Simple Approach using Linear Rules	78

5.3.2	Genetic Algorithms in Edge Detection using Cellular Automata	78
5.4	Forest Fire Model	79
5.5	Lotka-Volterra Equations	85
5.5.1	Approximating a Solution using a Cellular Automaton	87
6	Conclusion	91
7	Appendix	95
7.1	Time-Space Diagrams of Elementary Cellular Automata with Python .	95
	List of Figures	101
	List of Tables	103
	Bibliography	109
	Abstract	111
	Kurzfassung	115

CHAPTER 1 Introduction

It is always appealing, when simple processes yield complex outcomes. This phenomenon can often be observed in the realm of discrete dynamical systems. Many people are stunned by the beauty of fractals or by the moment, when the logistic equation slowly approaches the edge of chaos. Often this fascination simply stems from some kind of visual output derived from the behaviour of these models. In many cases, however, the real beauty is only revealed by understanding the underlying theory of these systems and the details of their construction.

The investigation of cellular automata started in the middle of the 20th century with an idea of JOHN VON NEUMANN. He wanted to show that self-replicating artificial systems are theoretically possible. These are deterministic systems, that - provided with the right instructions - build an exact duplicate of themselves. The first idea of NEUMANN were machines that are able to wander through a warehouse of parts, collect the needed parts and then build an exact copy of themselves. Later - following a suggestion of his friend STANISLAW ULAM - he invented a more abstract and elegant way of proofing the existence of such machines.

His new model used an infinite chequerboard of cells which can take several states and have a certain number of neighbours in a predefined neighbourhood. The states of the cells in that neighbourhood influence the state of the cell under consideration in the evolution of the system. He called these systems *cellular spaces*. In each time step, each cell has an explicit state, which is an element of a predefined state set and the cells change their state synchronously as a function of the states of the neighbouring cells in a predefined neighbourhood.

Cellular automata are therefore discrete dynamical systems, that are very simple to construct, but can yield very complex and self-organizing behaviour. It turned out that cellular automata can be used as models of natural computing and they have been subject to scientific investigations in a great variety of fields. While JOHN VON NEUMANN'S examinations took place at the edge of biology and computer science, cellular automata have been subject to scientific interest in a great variety of disciplines including mathematics, theoretical computer science, biology and physics but also in the social sciences and philosophy.

Apart from interesting theoretical properties, there are also many practical applications for cellular automata. Over the last decades many scientists were fascinated by the complexity that arises from such simple constructions. STEPHEN WOLFRAM dedicated a huge part of his academic life to the analysis of cellular automata and even proposed a *New Kind of Science* based on the analysis of these models. Many other scientists contributed to the analysis of these systems and nowadays we dispose of a large number of methods to describe a great variety of aspects of the complex behaviour of these *Miracles of Simplicity*.

This thesis is a survey which presents the current state of research in the realm of cellular automata, gives insight into some general theoretical results and finally provides an overview of selected applications. The focus, however, lies on theoretical properties of cellular automata. The remainder of this thesis is organized as follows:

First, we will provide basic definitions and concepts which will be used throughout this thesis. These definitions and concepts are an inevitable basis to understand the theoretical results described in the next chapter, where we will discuss the most important theoretical findings within this field. The next section treats a special class of one dimensional systems, which are called elementary cellular automata. Finally another theoretical section treats the two dimensional case. In the concluding chapter we describe selected applications of cellular automata in cryptography, digital image processing and within the simulation of complex systems.

1.1 Definition

In the following section we will give a basic definition and will explain the basic constituent parts in more detail. The definition is mainly based on [8] and [11]. The cellular automaton $\mathcal{A}$ is defined by:

$$\mathcal{A} = (\mathcal{S}, \mathcal{N}, \mathcal{F}, \mathcal{T})$$

where $\mathcal{S}$ is a finite set of possible states $s \in \mathcal{S}$ for each cell, $\mathcal{N}$ the local neighbourhood of a cell, $\mathcal{F}$ the transition function of the automaton and $\mathcal{T}$ the underlying topology, which constitutes the universe of the system.

1.1.1 Topology

Cellular automata have a finite or infinite d -dimensional rectangular grid as their underlying topology, that is, the universe in which they live and evolve. Let $d \in \mathbb{N}$, then we call $\mathbb{Z}^d$ a d -dimensional cellular space. We call the elements of $\mathbb{Z}^d$ the cells and we can refer to these cells by their coordinates on the grid.

In the case of $\mathbb{Z}^1$, we have the possibility to give a two dimensional representation of the evolution of the automaton through time, by adding a time axis as the second axis and drawing the states of the automaton in discrete time steps under each other. We will see interesting examples of such representations in the next section. These automata are called one dimensional cellular automata, referring to the one dimensional grid they live on. This class has been extensively researched and classified by STEPHEN WOLFRAM and others.

In the case of $\mathbb{Z}^2$, the dynamic visual representation through time is only possible by drawing a new picture on the grid every time step. An interesting and well studied example of a two dimensional cellular automaton is the so-called *Game of Life*, which will be discussed later on.

1.1.2 Neighbourhood

Every cell in the universe has a certain neighbourhood. In [8], a basic definition of *neighbourhood* is given:

Definition 1. Neighbourhood in $\mathbb{Z}^d$: Let $N = (x_1, x_2, \dots, x_n)$ be a vector of n distinct elements of $\mathbb{Z}^d$. Then the neighbours of a cell at location $x \in \mathbb{Z}^d$ are the n cells at locations $x + x_i$ for $i = 1, 2, \dots, n$.

The concept of a neighbourhood is essential for cellular automata, as the evolution of a cell's state through time depends on the state of the cells in its local neighbourhood. In the one dimensional case, the neighbourhood under consideration is referenced by a radius $r \in \mathbb{N}$. In the two dimensional case, there are many possible neighbourhoods for a cell to take into consideration, but two of them are the most important: the *Von-Neumann-Neighbourhood* and the *Moore-Neighbourhood*.

The Von-Neumann-Neighbourhood

Following [8] we use the Manhattan-Metric to define the Von-Neumann-Neighbourhood.

Definition 2. Manhattan-Metric: Let $p = (p_1, p_2, \dots, p_n)$ and $q = (q_1, q_2, \dots, q_n) \in \mathbb{R}^n$. The Manhattan- or Taxicab-Metric d_1 is defined by:

$$d_1(p, q) := \sum_{i=1}^n |p_i - q_i|$$

Definition 3. Von-Neumann-Neighbourhood: Consider $\mathbb{Z}^2$ (a square lattice in two dimensional Euclidean space). Let $p \in \mathbb{Z}^2$. For $n \in \mathbb{N}$ and $i \in \{1, 2, \dots, n\}$ let $q_i \in \mathbb{Z}^2$. All cells q_i for which $d_1(p, q_i) \leq r$ belong to the Von-Neumann-Neighbourhood of range r of p .

To be more precise, consider $\mathbb{Z}^2$ and let (x_0, y_0) be the Cartesian coordinates of x . Then the Von-Neumann-Neighbourhood $N_{(x_0, y_0)}^v$ of x is:

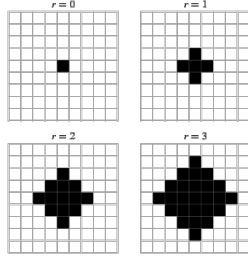


Figure 1.1: Von-Neumann[35]

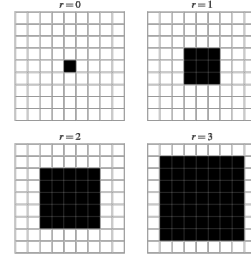


Figure 1.2: Moore[36]

$$N_{(x_0, y_0)}^v := \{(x, y) : |x - x_0| + |y - y_0| \leq r\}$$

If $r = 1$, we see that each cell in the location x of $\mathbb{Z}^d$ has $2d + 1$ neighbours, which are the cell itself and all cells in $x \pm e_i$, where e_i is the i -th unit vector of $\mathbb{Z}^d$. [8]

In Figure 1.1, we see examples of Von-Neumann-Neighbourhoods of range 1 to 4.

The Moore-Neighbourhood

The Moore-Neighbourhood can be defined using the Chebyshev-Metric [8].

Definition 4. Chebyshev-Metric: Let $p = (p_1, p_2, \dots, p_n)$ and $q = (q_1, q_2, \dots, q_n) \in \mathbb{R}^n$. The Chebyshev-Metric $d_{Chebyshev}$ is defined by:

$$d_{Chebyshev}(p, q) := \max_i (|p_i - q_i|)$$

Definition 5. Moore-Neighbourhood: Let $p \in \mathbb{Z}^d$. The Moore-Neighbourhood of range r of p is p itself and all cells q_i for which $d_{Chebyshev}(p, q_i) \leq r$.

In $\mathbb{Z}^2$, the Moore-Neighbourhood of range r of (x_0, y_0) is therefore defined by:

$$N_{(x_0, y_0)}^M = \{(x, y) : |x - x_0| \leq r, |y - y_0| \leq r\}$$

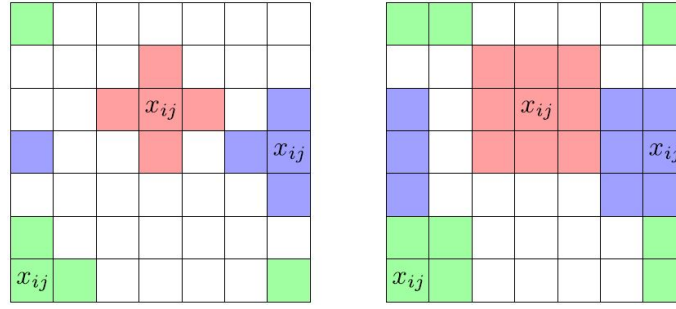


Figure 1.3: Periodic boundary conditions[26]

In Figure 1.2, samples of Moore-Neighbourhoods in two dimensional space of range $r \in [1, 4]$ are given.

Boundary Conditions

When using a finite grid, it is important to define what happens if a cell or one of its neighbours is located at the edge of the universe. Finite and periodic boundary conditions can be used. When using fixed boundary conditions, every cell which does not lie on the grid is assigned a fixed state. Periodic boundary conditions are achieved by modelling the universe as a torus. The left neighbours of the cells on the left side of the grid are the cells on the right side of the grid.

The choice of the boundary condition has important implications on the behaviour of the cellular automaton. SALMAN offers an analysis on the influence of various boundary conditions on the behaviour of one dimensional cellular automata and their implications for applications, such as random number generation and cryptography [23]. Figure 1.3 shows the Von-Neumann-Neighbourhood and the Moore-Neighbourhood of a cell x_{ij} under periodic boundary conditions. The cells which belong to the respective neighbourhood are coloured respectively.

1.1.3 State Set or Alphabet

Cells can take one of several states $s \in \mathcal{S}$ in each time step. The state set is finite. For $n \in \mathbb{N}$ it contains the discrete states $s_1, s_2, \dots, s_n$. In the most simple but non-trivial case the state set is given by $\{0, 1\}$. Sometimes, especially in the context of computer science, the state set is also called the *alphabet* and the states are called the *letters*. We call the function

$$c : \mathbb{Z}^d \longrightarrow S$$

the **configuration** of the cellular automaton. It assigns to each cell on the grid an element of the state set, hence a state. Considering $\mathbb{Z}^1$, we can refer to each cell by its one dimensional coordinate i and hence the state of cell $i \in \mathbb{Z}^1$ is $c(i)$.

1.1.4 Transition Functions

Local Update Rule

For a neighbourhood containing m cells, the local update rule is a function

$$f : S^m \longrightarrow S.$$

If $s_1, \dots, s_m$ are the states of the neighbours of a cell at time t , then $f(s_1, \dots, s_m)$ is the state configuration at $t + 1$.

Global Transition Function

Since all cells on the grid use the same update rule, which is applied to all cells at the same time, we can also define a global transition function, which reflects the change of a given configuration in one time step. For all $n \in \mathbb{Z}^d$ the global change from configuration c to c' is reflected by:

$$c'(n) = f[c(n + n_1), c(n + n_2), \dots, c(n + n_m)]$$

Definition 6. Global Transition Function: We call the transformation $c \longrightarrow c'$ the global transition function of the cellular automaton. If we denote a function from set

A into set B as B^A , then the global transition function can be described as

$$G : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}.$$

CHAPTER 2 Basic Properties of Cellular Automata

In this chapter we will discuss certain important theoretical concepts in the realm of cellular automata. This chapter aims to conclude with one of the most important theorems in this field, the Moore-Myhill Theorem or Garden-of-Eden Theorem. To formulate this theorem, we need to discuss some theoretical background and give definitions of certain basic concepts. These basic concepts can be found in many sources, but we mainly refer to [11] and [10].

2.1 Global Transition Functions, Orbits, Patterns and Orphans

In the theoretical analysis of cellular automata, the global transition function constitutes the main object of interest. Usually, this function is applied repeatedly and this iteration produces the dynamic evolution of the system in time. With a given starting configuration c we can denote this iteration process as

$$c \rightarrow G(c) \rightarrow G^2(c) \rightarrow G^3(c) \rightarrow \dots$$

and we call the sequence

$$c, G(c), G^2(c), G^3(c), \dots$$

the orbit of c or $\text{orb}(c)$. We can identify several possible orbits. According to [11] a configuration c is called

- a **fixed point** of G if $G(c) = c$.

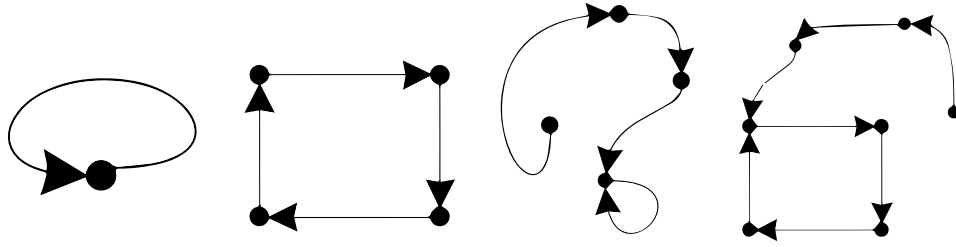


Figure 2.1: Possible Orbits: l. to r: fixed point, temporally periodic, eventually fixed, eventually temporally periodic [22]

- **temporally periodic** if $G^t(c) = c$ for some $t \in \mathbb{Z}$, which is then called the period of c .
- **eventually fixed** if $\exists t \in \mathbb{N} : G^{t+1}(c) = G^t(c)$. Then $G^n(c)$ is a fixed point for some n .
- **eventually temporally periodic** if $\exists t \in \mathbb{N}$ and $n \in \mathbb{N}$ so that : $G^{n+t}(c) = G^n(c)$. Here, $G^n(c)$ is periodic for some n .

Figure 2.1 shows illustrations for the different orbits.

We will now define several important concepts specifically related to cellular automata.

Definition 7. Let $\mathcal{A}$ be a cellular automaton on $\mathbb{Z}^d$ with state set $\mathcal{S}$. A **pattern**

$$p = (D, g)$$

is a partial configuration where we call $D \subseteq \mathbb{Z}^d$ the *domain* of p and $g : D \rightarrow \mathcal{S}$ the mapping which assigns a state to every cell in the domain.

We call p a **finite pattern** if D is a finite set. If the domain of p is the entire space, that is $D = \mathbb{Z}^d$, then this pattern is a *configuration* as defined above.

Furthermore, we call $p_1 = (D_1, g_1)$ a **subpattern** of $p_2 = (D_2, g_2)$ if $D_1 \subseteq D_2$ and

$$g_1(n) = g_2(n) \forall n \in D_1.$$

We say that p_2 contains a **copy** of p_1 if p_1 is a subpattern of p_2 .

Finally p_1 and p_2 are called **disjoint** if their respective domains are disjoint, that is $D_1 \cap D_2 = \{\}$.

2.2 Sequences and Convergence of Configurations

The contents of this section are mainly based on [11] and [10]. We denoted a configuration of a cellular automaton as a function in $S^{\mathbb{Z}^d}$, which can be given a compact topology under which all cellular automaton functions are continuous [10].

Definition 8. Convergence: We consider an infinite sequence of configurations

$$(c_1, c_2, c_3 \dots) \text{ where } c_i \in S^{\mathbb{Z}^d} \forall i$$

We define the convergence of the sequence and its limit $c \in S^{\mathbb{Z}^d}$:

$$\lim_{i \rightarrow \infty} c_i = c \Leftrightarrow \forall n \in \mathbb{Z}^d : \exists k \in \mathbb{N} : \forall i \geq k : c_i(n) = c(n)$$

This means that we can pick an arbitrary cell and go through the change of its state through time. If the limit exists, then, from some time step onwards, no further state change occurs which holds for every cell. The limit $\lim_{i \rightarrow \infty} c_i = c$ is this 'final' configuration.

Definition 9. Subsequence: We call a sequence $(c_{n_k})_{k \in \mathbb{N}}$ a subsequence of $(c_n)_{n \in \mathbb{N}}$ where $n_1 < n_2 < n_3 < \dots$ is a strict monotonic sequence. Hence, a subsequence is obtained by cutting a finite or infinite number of elements out of the original sequence. Note that every sequence is a subsequence of itself.

Proposition 1. Every sequence of configurations has a converging subsequence.

Proof: Can be found in [11].

Proposition 2. Let G be the global transition function of a cellular automaton and $(c_1, c_2, \dots)$ a converging sequence of configurations. Then also the sequence $G(c_1), G(c_2), \dots$ converges and if

$$c = \lim_{i \rightarrow \infty} c_i$$

then

$$\lim_{i \rightarrow \infty} G(c_i) = G(c).$$

Proof: Can be found in [11].

2.3 Quiescent State, S-Support and S-Finite Configurations

Definition 10. Quiescent State: A state $s \in \mathcal{S}$ is called quiescent, if it satisfies

$$f(s, s, \dots, s) = s$$

which means that, if all neighbours of a cell are in this quiescent state, the cell itself becomes quiescent when f is applied. If we observe a configuration c_q in which every cell is in the quiescent state then we call c_q the quiescent configuration.

Definition 11. S-Support: Let $s \in \mathcal{S}$. We call the set

$$\text{supp}_s(c) = \{n \in \mathbb{Z}^d \mid c(n) \neq s\}$$

the s -support of configuration $c \in \mathcal{S}^{\mathbb{Z}^d}$. This means we refer to all cells which are not in state s .

Definition 12. S-Finite: We call a configuration c s -finite if $\text{supp}_s(c)$ is finite. The set of all configurations c which are s -finite is denoted as

$$\mathcal{F}_s(d, \mathcal{S}) = \{c \in \mathcal{S}^{\mathbb{Z}^d} \mid c \text{ is } s\text{-finite}\}.$$

Let q be a quiescent state. Sometimes, a configuration is simply called *finite* if

$$\{n \in \mathbb{Z}^d \mid c(n) \neq q\}$$

is finite. This means that the number of cells which are not in the quiescent state is finite.

2.4 Basic Properties of Transition Functions

Definition 13. Recall that we call a function $f : A \longrightarrow B$ injective, if

$$\forall b \in B : |f^{-1}(b)| \leq 1,$$

we call it surjective if

$$\forall b \in B : |f^{-1}(b)| \geq 1$$

and bijective, if it is both injective and surjective.

Definition 14. Transition Function Properties: We call a cellular automaton injective, surjective or bijective, respectively, if its global transition function has the respective property.

Definition 15. Reversible Cellular Automata: We call a global transition function G of a cellular automaton *reversible*, if it is bijective and if G^{-1} (its inverse function) is also a global transition function. Finally we call a cellular automaton $\mathcal{A}$ reversible, if its global transition function is reversible. We call the automaton with global transition function G^{-1} the *inverse automaton* of $\mathcal{A}$. It follows directly that

$$G \circ G^{-1} = G^{-1} \circ G = id$$

where id denotes the identity function.

Note that every reversible cellular automaton has to be bijective by definition.

Definition 16. Finite Configurations: The subset of $S^{\mathbb{Z}^d}$ (all configurations) which contains only finite configurations is denoted as $\mathcal{C}_F$.

Since the quiescent configuration q is stable, q -finite configurations remain q -finite in the further evolution of the automaton. Therefore G_F is a function $\mathcal{C}_F \rightarrow \mathcal{C}_F$.

Definition 17. Periodic Configurations: A periodic configuration is a configuration that is invariant under d linearly independent translations. For $i = 1, 2, \dots, d$ let e_i be the i -th unit vector in $\mathbb{Z}^d$. A configuration c is periodic if

$$\forall x \in \mathbb{Z}^d : \exists t_1, t_2, \dots, t_d : c(x) = c(x + t_i e_i).$$

We denote $\mathcal{C}_P(d, S)$ or just $\mathcal{C}_P$ for the set of periodic configurations. Since periodic configurations are preserved in the evolution of cellular automata the restriction of G on only the periodic configurations, that is G_P , is a function $\mathcal{C}_P \rightarrow \mathcal{C}_P$.

Finite and periodic configurations are often used to simulate the behaviour of cellular automata on computers. Periodic configurations are often referred to as *periodic boundary conditions* on a finite cellular array. For a two dimensional cellular automaton this is equivalent to running the cellular automaton on a torus [10].

Definition 18. Orphan: We call a finite pattern which has no preimage an *orphan*.

Definition 19. Garden-of-Eden Configuration: We call a configuration c of a cellular automaton $\mathcal{A}$ with global transition function G a *Garden-of-Eden configuration*, if

$$G^{-1}(c) = \{\}$$

Let c_g be a Garden-of-Eden configuration, then this definition simply means that

$$\forall c : f(c) \neq c_g.$$

Hence, a Garden-of-Eden configuration has no preimage. It is clear that if and only if a cellular automaton is not surjective, it has a Garden-of-Eden configuration.

Proposition 3. Every configuration that contains an orphan is a Garden-of-Eden configuration.

Proof: If in a given configuration any subpattern has no preimage, there cannot be a global preimage of the configuration. If so, there would be a preimage for the subpattern, contradicting the assumption that it is an orphan.

□

Also the inverse is true:

Proposition 4. Every Garden-of-Eden configuration has a subpattern that is an orphan.

Proof: This proof can be found in [11]. Let $c \in S^{\mathbb{Z}^d}$ be a Garden-of-Eden configuration. Recall that c is a pattern itself. Indirectly suppose that no subpattern of c is an orphan. We enumerate the elements of $\mathbb{Z}^d$ as $(r_1, r_2, \dots)$. For $j \in \mathbb{N}$ denote subpatterns of c as $D_j = \{r_1, r_2, \dots, r_j\}$. For every $j \in \mathbb{N}$, D_j is not an orphan, so we can find a configuration $c_j \in S^{\mathbb{Z}^d}$ so that $G(c_j) = c$ in domain D_j . But this means that the sequence $G(c_1), G(c_2), \dots$ converges to c . As every sequence has a converging subsequence $(c_{i_k})_{k \in \mathbb{N}}$ (Proposition 1), we can define $e := \lim_{k \rightarrow \infty} c_{i_k}$ where $e \in S^{\mathbb{Z}^d}$. Because G is continuous, we obtain

$$\lim_{k \rightarrow \infty} G(c_{i_k}) = G(e).$$

Since

$$\lim_{j \rightarrow \infty} G(c_{i_j}) = \lim_{i \rightarrow \infty} G(c_i) = c$$

we find that $G(e) = c$. Since c was a Garden-of-Eden configuration, this result contradicts our initial assumption that c is a Garden-of-Eden configuration. This proves that c cannot be a Garden-of-Eden configuration.

□

2.5 Moore-Myhill Theorem

Proposition 5. Let $n \in \mathbb{N}$. Let G be the global transition function of a cellular automaton with n cells. The following holds:

$$G \text{ not injective} \implies G \text{ not surjective}$$

Proof: Consider a one dimensional cellular automaton with two possible states and n cells in total. We can find a total of 2^n different configurations for this automaton. If the global transition function was bijective, it would map the set of configurations onto the 2^n possible configurations again. Suppose now that the global transition function is not injective, meaning that two different configurations c_1 and c_2 are mapped onto the same configuration c_0 . Then the global transition function only generates $2^n - 1$ different configurations. Hence c' , one out of the total 2^n configurations, cannot have a predecessor. The global transition function does not map any configuration onto c' . This implies that it is not surjective. □

In the above proposition, we let $n \in \mathbb{N}$, meaning that we only consider finite grid sizes. The Moore-Myhill Theorem, however, generates predictions on injectivity and surjectivity for an infinite array of cells. For this case another restriction on the function is needed.

Consider $\mathcal{C}_F$, which we defined as the set of all finite configurations of a cellular automaton. In a *finite* configuration all but a finite number of cells are in a quiescent state. Recall that we defined the function G_F as the global transition function restricted to finite configurations:

$$G_F : \mathcal{C}_F \longrightarrow \mathcal{C}_F$$

We defined Garden-of-Eden configurations as configurations without a preimage. This implies that if such a configuration exists, the cellular automaton is not surjective. If

given a finite set and a function from the set to itself, then the function is injective if and only if it is surjective. For global transition functions of cellular automata this is only true in one direction. Every injective cellular automaton is surjective [11]. But not every surjective automaton is automatically injective.

However, if we consider the restricted function G_F , then the following important theorem, known as the Moore-Myhill Theorem or Garden of Eden Theorem, holds:

Proposition 6. Moore-Myhill Theorem: Let $\mathcal{A}$ be a cellular automaton. Let $\mathcal{C}_F$ be the set of all finite configurations and let

$$G_F : \mathcal{C}_F \longrightarrow \mathcal{C}_F.$$

Then G_F is injective if and only if G is surjective. This also implies that the restricted transition functions G_F and G_P are surjective if and only if G_F is injective.

This is one of the most important results in the theoretical investigations of cellular automata and is also known as the **Garden-of-Eden Theorem**. This theorem connects injectivity and surjectivity of global transition functions of cellular automata.

Corollary 1. Let $\mathcal{A}$ be a cellular automaton with global transition function G . The following holds:

$$G \text{ injective} \implies G_F \text{ injective}$$

and

$$G \text{ injective} \implies G_P \text{ injective}$$

Proof: Trivial, since G_F and G_P are both subsets of G .

□

We can directly derive the following observation:

Corollary 2. Injective cellular automata are also surjective. It follows that injectivity, bijectivity and therefore reversibility are equivalent.

This result is remarkable, since it allows to theoretically predict Garden-of-Eden configurations in cellular automata when the non-injectivity of the global transition function is known. It also allows to predicate reversibility of cellular automata when only one function property is known. If for example G_F is surjective, it is also injective. It is hence bijective meaning that the cellular automaton is reversible. The question *if* a given cellular automaton transition function on $\mathbb{Z}^d$ is injective or surjective is algorithmically undecidable for $d \geq 2$ as shown by KARI in [9]. For one dimensional cellular automata, however, this question is decidable.

Corollary 3. Finally, we summarize the following relations:

- If G is injective then G_F and G_P are injective.
- If G_F or G_P is surjective then G is surjective.
- If G_P is injective then G_P is surjective.
- If G is injective then G_F is surjective.
- G_F injective $\iff G$ surjective (Moore-Myhill Theorem).

This has some interesting implications. For example it is easy to shown that the two dimensional cellular automaton *Game of Life* which we will discuss later in more detail is not surjective, since a configuration with all cells in state 0 and another configuration where all but one cell are in state 0 map to the same configuration when the global transition function is applied. It follows from the Moore-Myhill Theorem, that because G_F is not injective, Game of Life is not surjective. This means that orphans and Garden-of-Eden configurations must exist.

Note that this is a theoretical result, which does not allow to decide for a given configuration if it is a Garden-of-Eden configuration. It is very difficult to actually find such configurations as we will see later.

2.6 Linear Cellular Automata

A special class of cellular automata is called linear cellular automata. We will later present some examples of this type. Here we will give the formal definition.

We have seen that in the theoretical analysis of cellular automata, the global transition function is our main object of interest. If this function G is applied to a configuration c (which is itself a function which assigns a state to every element of the space), we obtain another configuration $c' = G(c)$. Very often we have $\mathbb{Z}^d$ as the space and $\mathbb{Z}_2$, the residue classes modulo 2, as the state space. Obviously $(\mathbb{Z}_2, +, \cdot)$ is a field. The following definition can be found in [2].

Let $\mathcal{G}$ be a group and let V be a vector space over a field $\mathbb{K}$. We can identify the set $V^{\mathcal{G}}$ (the set of all possible configurations) as a vector space over $\mathbb{K}$, where each vector represents a certain configuration. A linear cellular automaton is further defined as follows:

Definition 20. Linear Cellular Automaton: A cellular automaton with global transition function G over $\mathbb{Z}^d$ with state set S , that is,

$$G : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d},$$

is called *linear* if S itself is a vector space over a field $\mathbb{K}$ and if for all $c, c' \in S^{\mathbb{Z}^d}$ and $k \in \mathbb{K}$ the following holds:

$$G(c + c') = G(c) + G(c') \text{ and } G(kc) = kG(c)$$

We will see examples and some further discussion of this class in a later chapter.

2.7 Basic Concepts of Complexity Theory

Many theoretical questions in the realm of cellular automata can be formulated as *decision problems*. The following definition can be found in [6].

Definition 21. Decision Problems and Decidability: A decision problem is a problem where the solution for a given instance is either yes or no. A decision problem is further called **decidable**, if there is an algorithm capable of deciding the problem in finite time.

The questions that arise from decision problems and their decidability play an important role in computer science and related mathematical fields. Therefore they are also important in the theory of cellular automata. Such a question might, for example, be whether a given cellular automata will ever reach a certain configuration or not.

The set of decidable problems is classified in so-called *complexity classes*, by the work which is needed to solve them. There are many complexity classes, but the most important is called *P*. P-hard Problems can be solved in polynomial time.

Definition 22. Polynomial Time: An algorithm is of polynomial time if there is a $k \in \mathbb{N}$ so that it takes no longer than n^k steps to solve a problem of length $n \in \mathbb{N}$.

Example: Consider a person cutting the lawn. We can formulate the length of the input problem n as the area of the lawn. If the area doubles, the time it takes to complete doubles as well. The optimal algorithm for lawn-cutting has a runtime of $\mathcal{O}(n)$ and is therefore of polynomial time. If we consider a square-lawn and describe the input-length as the side of the lawn, then the algorithm has a runtime of $\mathcal{O}(n^2)$.

If the given problem also belongs to *P* it is called P-complete. To prove that a given problem is P-hard, it is sufficient to reduce a P-complete problem to it [6].

CHAPTER 3 --- One Dimensional Cellular Automata

In the following section, we will discuss one dimensional Elementary cellular automata. These have been studied intensively by STEPHEN WOLFRAM and other authors. We will start with an example and later discuss some general properties of this class.

3.1 Rule Number as Description for the Local Function

In the following section, we present the standard classification scheme as introduced by WOLFRAM in [39].

Definition 23. Elementary Cellular Automata: Let i be a cell on a one dimensional lattice $\mathcal{T}$. Let $\mathcal{S} = \{0, 1\}$ and $s \in \mathcal{S}$. We consider the neighbourhood $\mathcal{N} = \{i - 1, i, i + 1\}$. Let s_i^t be the state of cell i at time step t . For a given local rule $\mathcal{F}$, we obtain

$$s_i^{t+1} = \mathcal{F}(s_{i-1}^t, s_i^t, s_{i+1}^t)$$

and we call $\mathcal{A} = (\mathcal{S}, \mathcal{N}, \mathcal{F}, \mathcal{T})$ an **Elementary Cellular Automaton**.

Since every cell in our 3-cells neighborhood (including the cell itself) can take 2 states, we obtain $2^3 = 8$ different configurations. Every configuration can be mapped to either 0 or 1, so we can obtain $2^8 = 256$ different elementary cellular automata. These were studied by WOLFRAM in [43], [45] and [44]. He also invented a notation to identify the cellular automata based on the update rule. Therefore we consider the images of

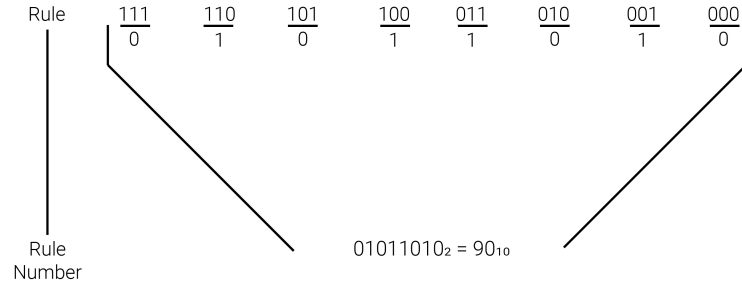


Figure 3.1: Wolfram Classification Scheme[41]

all possible neighbourhood configurations under f :

$$f(111)f(110)f(101)f(100)f(011)f(010)f(001)f(000)$$

Every function value is either 0 or 1, since our state set contains only these two numbers. Hence, every rule can be identified with the function values in the above given order. We obtain an 8-bit sequence, which can easily be converted into a base 10 integer in the interval $[0, 255]$. This base 10 number is then called the **rule number**. Figure 3.1 visualizes this idea. As already mentioned, we can depict a visual representation of the behaviour of the automaton as a time-space diagram. The Python script in Appendix 1 calculates and prints time-space diagrams for all 256 rules and was written to generate the images in this thesis. All images of elementary cellular automata in this thesis were produced using this script. It makes use of some open-source libraries like Numpy for matrix operations and Matplotlib to produce a visual output of the calculated matrix.

3.2 Examples

Figures 3.2-3.7 show time-space diagrams of selected elementary cellular automata.

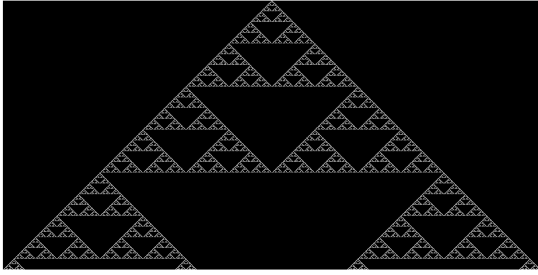


Figure 3.2: Rule 90

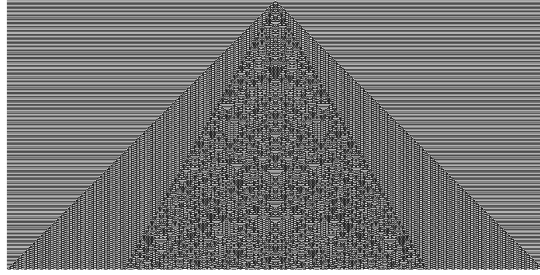


Figure 3.3: Rule 73

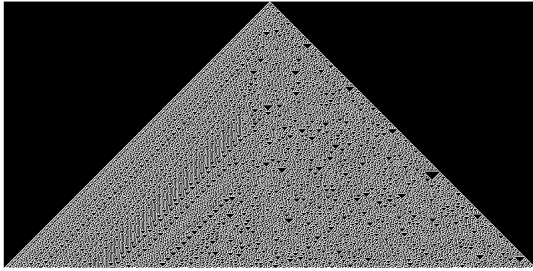


Figure 3.4: Rule 30

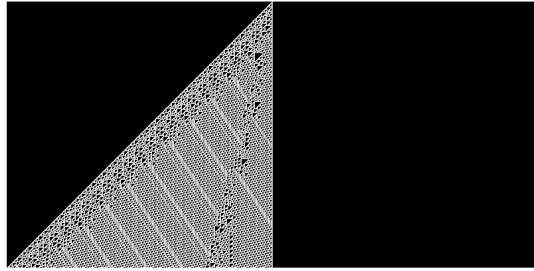


Figure 3.5: Rule 110

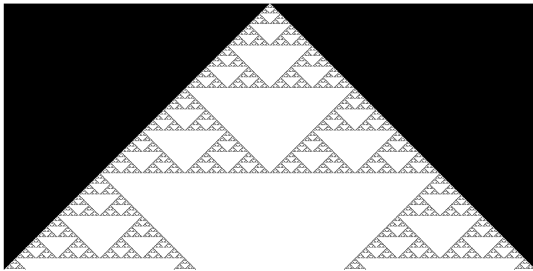


Figure 3.6: Rule 182

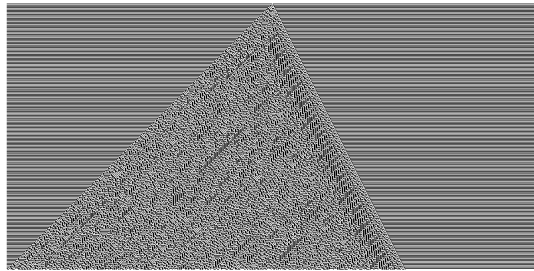


Figure 3.7: Rule 101

3.3 Fractal Structures

We will now analyse the example provided in Figure 3.2 in more detail.

Consider a row of n cells, hence a subset of $\mathbb{Z}^1$. We refer to each cell as $i \in \{1, 2, \dots, n\}$.

The state set is $\{0, 1\}$. The neighbours of a cell i are only $i - 1$ and $i + 1$, which are the cells left and right of cell i . Let s_i^t be the state of the i -th cell at time step t . Our transition function is given by:

$$s_i^{t+1} = s_{i-1}^t + s_{i+1}^t \mod 2$$

This definition is obviously equivalent to

$$s_i^{t+1} = s_{i-1}^t \vee s_{i+1}^t$$

where $\vee$ represents the exclusive or (XOR).

It is easy to see, that our local rule function maps onto the state set again, since the set of residue classes modulo 2 contains only 0 and 1. We need to provide an initial configuration of the system, which are the states of the cells at the initial time step. To represent our automaton and its evolution through time visually, we colour each cell with a state of 1 in white, and a cell with state 0 in black. We give a depiction of the time-space evolution of the system, by drawing the configuration for each time step below each other.

For the first example, our initial configuration is such that every cell has state 0 except of the middle one, which has state 1. What we get is a mod 2 reduced version of Pascal's Triangle, that is, the reduction of the coefficients of x^i in the series expansion of $(1+x)^n$. The even coefficients in the triangle are our cells with state 0 and the odd coefficients are cells with state 1. Figure 3.8 shows our example after 5 time steps. Figure 3.9 after 50 time steps and Figure 3.10 after 100 steps.

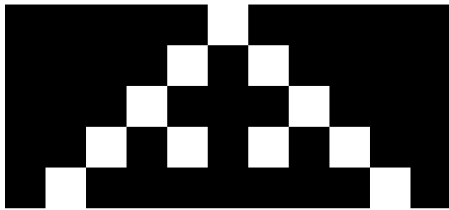


Figure 3.8: Rule 90, 5 time steps

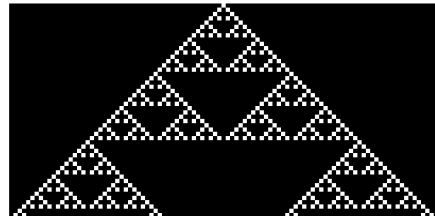


Figure 3.9: Rule 90, 50 time steps

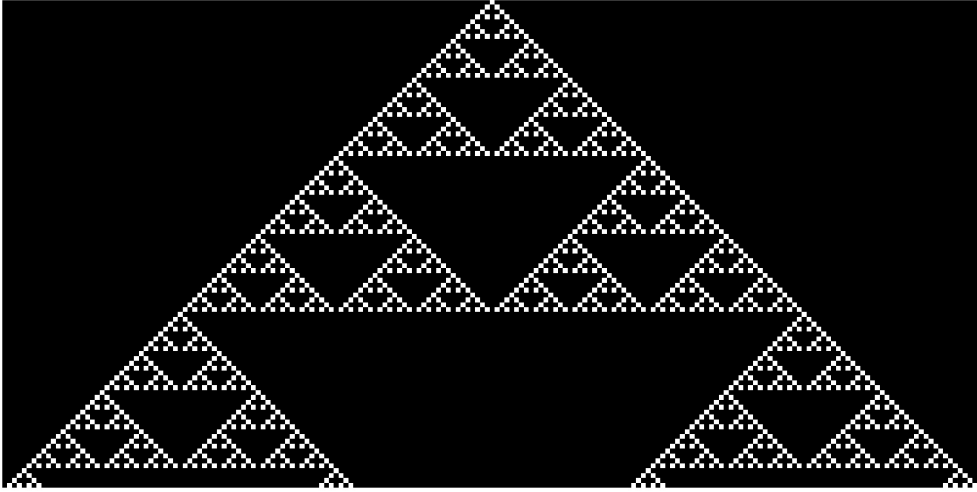


Figure 3.10: Rule 90, 100 time steps

In these diagrams, we see that this simple cellular automaton, represented in this manner, produces an approximation of the Sierpinski Triangle in the large time limit, with the same structure and fractal dimension of $\log_2 3$. This means that the generated structure shows self-similarity or *scale invariance*. Figure 3.11 shows the geometric construction of the Sierpinski triangle.

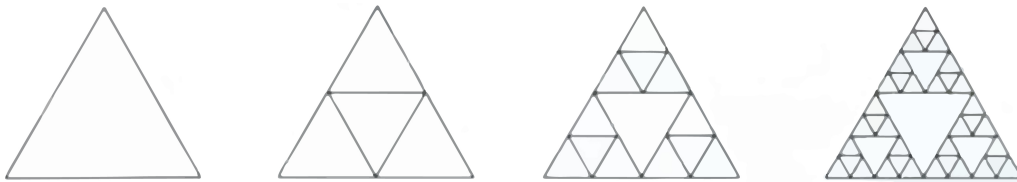


Figure 3.11: Geometric construction of rule 90[40]

It is, however, very important to note that the image we see is not the cellular automaton itself, but its evolution in time.

Another fractal structure is produced by the modulo 2 reduction of the coefficients of

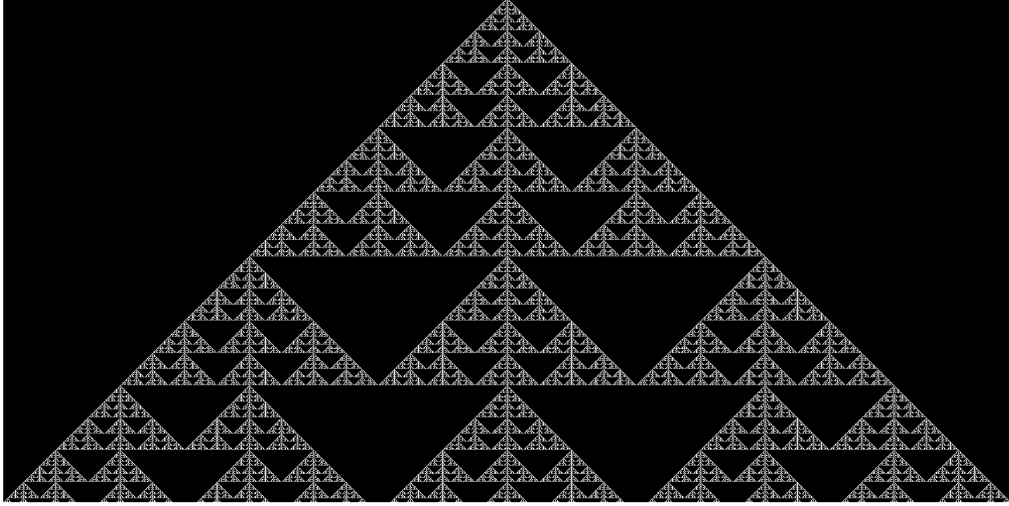


Figure 3.12: Rule 150, 1000 time steps

x^i in the series expansion of $(x^2 + x + 1)^n$. The transition function is given by:

$$s_i^{t+1} = s_{i-1}^t + s_i^t + s_{i+1}^t \mod 2$$

This function has the boolean form:

$$s_i^{t+1} = s_{i-1}^t \vee s_i^t \vee s_{i+1}^t$$

The decimal rule number is 150. If we calculate a depiction of the time-space diagram produced by this cellular automaton in 1000 time steps with a single starting cell, we obtain the image of Figure 3.12.

Like the Sierpinski-Triangle, the curve can also be constructed geometrically as seen in Figure 3.13.

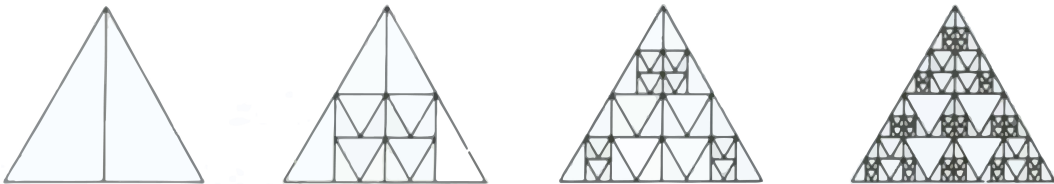


Figure 3.13: Geometric construction of rule 150[40]

This is a self-similar structure with fractal dimension $\log_2 2\varphi$, where $\varphi = \frac{(1+\sqrt{5})}{2}$. The fact, that such simple rules can yield such non-trivial behaviour, has undoubtedly contributed to the popularity of elementary cellular automata both within the scientific community as well as in the general public.

3.3.1 Random Initial Conditions

Now consider rule 90 with random initial conditions. Still, we can observe some patterns in the structure.

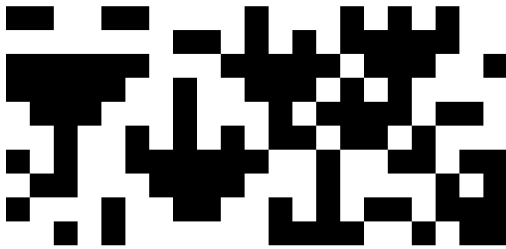


Figure 3.14: Rule 90, 5 time steps



Figure 3.15: Rule 90, 50 time steps

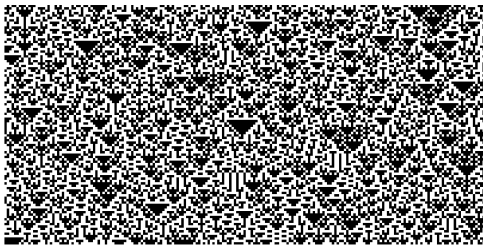


Figure 3.16: Rule 90, 100 time steps



Figure 3.17: Conus textile¹

Interestingly, these structures are reminiscent of the shell of Conus Textile, a sea snail found in several oceans. It is often the case that cellular automata produce structures similar to those in nature.

¹Richard Ling, https://commons.wikimedia.org/wiki/File:Textile_cone.JPG

3.4 Rule Classification Scheme

In [45] WOLFRAM experimented with elementary cellular automata and found, that "many (perhaps all) cellular automata fall into four basic classes" [45]. Within each class, we can observe similar behaviour of the automaton. WOLFRAM notes that, due to the small number of classes, a "considerable universality in the qualitative behavior of cellular automata" [45] can be observed. This means that the qualitative behaviour of the system is independent of its details of construction. He concludes that "thus complex physical and biological systems may lie in the same universality classes as the idealized mathematical models provided by cellular automata. Knowledge of cellular automaton behavior may then yield rather general results on the behavior of complex natural systems" [45]. He identified the following classes:

1. **Class 1:** The evolution leads to a homogeneous state almost regardless of the initial configuration.
2. **Class 2:** The evolution leads to periodic patterns for almost all initial configurations.
3. **Class 3:** The evolution leads into chaotic behaviour for almost all initial configurations.
4. **Class 4:** The evolution leads to complex structures.

In Figures 3.18 - 3.21 we will now provide example representations for all four classes and will then describe the properties of the four classes in more detail. All examples are calculated with random initial conditions, but the number of time steps was adjusted to give a good depiction of the behaviour.



Figure 3.18: Class 1: Rule 160

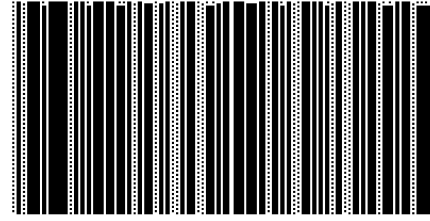


Figure 3.19: Class 2: Rule 108

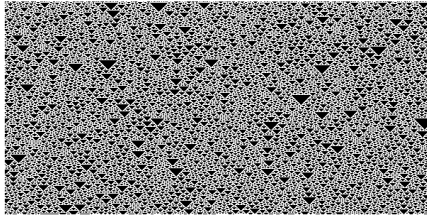


Figure 3.20: Class 3: Rule 126

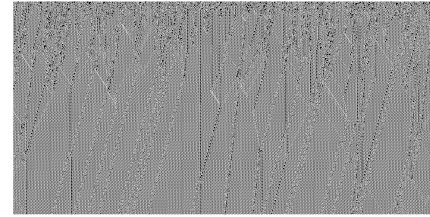


Figure 3.21: Class 4: Rule 110

It turns out that many properties of the evolution of elementary cellular automata only depend on the class in which they lie and not on the actual initial condition or exact details of their evolution. WOLFRAM states in [41] that "Such universality is analogous, though probably not mathematically related, to the universality found in the equilibrium statistical mechanics of critical phenomena. In that case many systems with quite different detailed construction are found to lie in classes with critical exponents that depend only on general, primarily geometrical features of the systems and not on their detailed construction." [41]

We will now describe the four classes in more detail, according to WOLFRAM in [45] and [41].

3.4.1 Elementary Class 1 Automata

Class 1 automata tend to a static state after a finite number of iterations. As we can see in Figure 3.18, only after a few iterations, all cells are in state 0. It is impossible

that this state will ever change again. All information coded in the initial state of the system is therefore lost.

3.4.2 Elementary Class 2 Automata

Elementary class 2 automata serve as filters which select certain properties of their initial state and 'filter out' others in their evolution while they generate rather simple structures. These structures can be stable or periodic. For example, elementary cellular automata with rule number 8, 24, 40 or 56 generate stable structures. On the other hand, the above given example of rule 108 is an example of an Elementary Class 2 automata that generates a periodic structure with a small period.

3.4.3 Elementary Class 3 Automata

In contrast to Class 2, where we observe a rather stable behaviour of the evolution of the automaton which lead to periodic structures, the evolution of elementary Class 3 automata leads to aperiodic (chaotic) structures. WOLFRAM conducted statistical analysis on the behaviour of class 3 automata and found that after a sufficient number of time steps, the statistical properties of the patterns generated by members of this class are almost the same for all initial conditions. More precise, he found that given a state set $\mathcal{S}$ (e.g. $= \{0, 1\}$) the density of non-zero sites often tends to a non-zero value typically close to $\frac{1}{|\mathcal{S}|}$, where $|\mathcal{S}|$ simply denotes the cardinality of the state set [43]. This means that the observed states are often equally distributed. Irregularity produced by class 3 automata may however vary depending on the applied rule. It can range from highly irregular structures to rather regular structures. Often, we observe the emergence of triangular fields, in which all sites have the same value. According to WOLFRAM in [43] "These clearings occur when a 'fluctuation' in which a sequence of consequence of consecutive sites have the same value, is progressively destroyed by the effects of other sites."

3.4.4 Elementary Class 4 Automata

Class 4 cellular automata have a very special behaviour. These automata produce propagating structures which are sometimes long-lived. In [43] WOLFRAM supposed that elementary class 4 automata may be computationally universal hence Turing complete. He continued to work on this conjecture and in the meantime this universality could be proven for rule 110. The proof was found by COOK while he worked with WOLFRAM at Wolfram Research Inc. assisting the latter in his works for [39]. COOK published the proof in [4].

3.5 Further Discussion of the 4 Classes

As outlined in [8], the above discussed classification by WOLFRAM is mathematically vague and not formalized. They are mostly motivated by statistical analysis of the time-space evolution of the automata and WOLFRAM did not propose any formal way of determining in which class an automaton lies. The question, if there actually is a formal way to decide on this matter was addressed by CULIK and YU. To answer this question, a more formalized definition of the four classes was needed.

3.5.1 Formalized Approach by CULIK and YU

The formal approach to the four classes proposed in [5] by CULIK and YU uses some of the definitions provided above. The four classes are formalized in the following way (see [5], [8] or [28] for reference):

- **(CY1)** All finite configurations evolve into the quiescent configuration Q , which means:

$$\forall c \in \mathcal{C}_F : \exists n \geq 1 : G^n(c) = Q$$

- **(CY2)** All finite configurations are eventually periodic, which means:

$$\forall c \in \mathcal{C}_F : \exists m, n, m \neq n : G^m(c) = G^n(c)$$

- **(CY3)** There exists an algorithm that determines if a given finite configuration belongs to the orbit of another given finite configuration, which means that it is decidable for given $c, e \in \mathcal{C}_F$ whether $G^n(c) = e$ for some $n \geq 1$.
- **(CY4)** No restriction.

In [5], CULIK and YU could show that it is undecidable to which class a given cellular automata belongs to.

3.6 Sequence of Numbers in Elementary Cellular Automata

3.6.1 The Jacobsthal Numbers

The following results can be found in [33]. Consider an elementary cellular automaton with the following transition function:

$$s_i^{t+1} = s_{i-1}^t + s_i^t + s_{i-1}^t \cdot s_{i+1}^t + s_{i-1}^t \cdot s_i^t \cdot s_{i+1}^t \mod 2$$

It has the following boolean form:

$$s_i^{t+1} = s_{i-1}^t \vee (s_i^t \vee (s_{i-1}^t \wedge s_{i+1}^t))$$

The rule name of this function in the classification scheme of WOLFRAM is rule 28. We use our Python script to calculate the first 10 time steps with a single cell starting condition:

```
python [filename].py -r 28 -t 10 -s 0
```



Figure 3.22: Rule 28

This generates the image of Figure 3.22.

We could interpret the white cells as 1 and the black cells as 0 and obtain a sequence of ten binaries (one for each line). These are:

```
00000000001
000000000011
0000000000101
00000000001011
000000000010101
0000000000101011
00000000001010101
000000000010101011
0000000000101010101
00000000001010101011
```

Converted to base 10, this is the following sequence: 1 3 5 11 21 43 85 171 341 683, which is known as the **Jacobsthal Numbers**, OEIS A001045. These can also be obtained by the following recurrence:

$$J_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ J_{n-1} + 2J_{n-2} & n > 1 \end{cases}$$

The numbers can also be calculated using the following closed form expression:

$$J_n = \frac{2^n - (-1)^n}{3}$$

3.6.2 The Mersenne Numbers

In fact, many elementary cellular automata produce well known integer sequences in this manner. Another example is rule 220. It is given by the following definition:

$$s_i^{t+1} = s_{i-1}^t + s_i^t + s_{i-1}^t \cdot s_i^t + s_{i-1}^t \cdot s_{i+1}^t + s_{i-1}^t \cdot s_i^t \cdot s_{i+1}^t \mod 2$$

It has the following boolean form:

$$s_i^{t+1} = s_{i-1}^t \vee (s_{i-1}^t \wedge \neg(s_{i+1}^t))$$

The first ten time steps produce the image in Figure 3.23.



Figure 3.23: Rule 220

By using the same method as above, we obtain the following sequence of binaries:

```

00000000001
000000000011
0000000000111
00000000001111
000000000011111
0000000000111111
00000000001111111
000000000011111111
0000000000111111111
00000000001111111111

```

Converted to base 10, this sequence is equivalent to: 1 3 7 15 31 63 127 255 511 1023, which are the Mersenne-Numbers, OEIS A000225. The n -th Mersenne-Number is obtained by the following recurrence:

$$M_n = 2^{n+1} - 1$$

Of course, this example seems trivial, since the Mersenne-Numbers have a rather simple construction in base 2. The n -th Mersenne number is simply represented by $(n+1)$ 1's in base 2. It is, however, an interesting fact that many elementary cellular automata produce well known integer sequences in this manner. Other examples include rule 50, which generates the sequence A002450 in OEIS, rule 54, which generates A118108 in OEIS, and many more.

In the paper *Generating functions for integer sequences defined by the evolution of cellular automata with even rule numbers* [14] as well as in a series of other studies MACFARLANE presents a unified treatment of generating functions for the evolution of elementary cellular automata from non-random initial conditions.

3.7 Computational Universality in Elementary Cellular Automata

An interesting and quite prominent result in the theory of elementary cellular automata is that rule 110 is Turing complete. WOLFRAM conjectured Turing completeness for all elementary class 4 automata in the 1980s. For rule 110, this conjecture could be proven by COOK in [4]. We will now present the idea of his proof preceded by some theoretical background.

3.7.1 Turing Machines

A Turing machine is a simple theoretical model of a computer.

Definition 24. Deterministic Turing Machine: The following definition is given in [11]. A deterministic Turing machine is given by the following components:

- a finite state set Q
- initial, accepting and rejecting states $q_0, q_a, q_r \in Q$, where q_a and q_r are both called halting states and $q_a \neq q_r$
- a finite tape alphabet Γ
- an input alphabet $\Sigma \subset \Gamma$
- a blank tape symbol $b \in \Gamma \setminus \Sigma$
- a translation function $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{-1, 1\}$. And $\forall \gamma \in \Gamma$, δ must fulfil $\delta(q_a, \gamma) = (q_a, \gamma, 1)$ and $\delta(q_r, \gamma) = (q_r, \gamma, 1)$.

This machine has a tape of infinite length, consisting of cells which can store a single letter of Γ . Like in elementary cellular automata, the position of a cell on the tape is

indexed by $\mathbb{Z}$. We can describe the content of the tape at a given time step by a function $t \in \Gamma^{\mathbb{Z}}$ where $t(i) \in \Gamma$ (the symbol in cell i) for any given $i \in \mathbb{Z}$.

The machine also has a control unit, which is a finite state automaton that moves along the tape and is capable of reading the symbol in a cell.

According to the Church-Turing Thesis, a Turing machine is capable of universal computation. This means that, for every algorithm, there is a theoretical implementation on a Turing machine [4]. This can also be used to define the concept of an algorithm.

In the following section, we will outline the universality proof for rule 110 given by Cook in [4].

3.7.2 Computational Universality of Rule 110

As mentioned in the beginning of this section, one of the most popular results of research in the realm of Elementary cellular automata is that rule 110 has shown to be Turing complete. This was already conjectured by WOLFRAM for all Class 4 Elementary cellular automata but could only be proven for rule 110 by now.

A universal system is a "completely general instruction-obeying mechanism" [40]. A system that is universal, which means that it is capable of universal computation, can run any program, that is, execute any algorithm. The input data for the program must be encoded in a way that the program can use. Also the output must be encoded in this way [4]. But how should Elementary cellular automata be computationally universal? In fact, cellular automata may be seen as computers that process data in their evolution through time. The input data is represented by the given initial condition. The concept of computational universality of a system means that suitable initial conditions can specify arbitrary algorithmic procedures. Hence, cellular automata may serve as a computer on which it is theoretically possible to evaluate any computable function [40]. To prove that a system $\mathcal{A}$ is computationally universal, it is sufficient to prove

that it is capable of emulating a Turing machine. This means that this machine can do everything a Turing machine could do. Since a Turing machine is computationally universal, this approach proves that $\mathcal{A}$ is too. The standard way of doing this is to present some sort of compiler, a machine that can take any Turing machine program as an input and produces a program for $\mathcal{A}$ as output. The restriction on the compiler is that it is required to finish within a predictable amount of time for any input program.

Cook constructed a way to prove computational universality for rule 110. The local update rule of rule 110 is the following:

$$s_i^{t+1} = s_i^t + s_{i+1}^t + s_i^t \cdot s_{i+1}^t + s_{i-1}^t \cdot s_i^t \cdot s_{i+1}^t \mod 2$$

It has the following boolean form:

$$s_i^{t+1} = (s_{i-1}^t \wedge \neg s_i^t) \vee (s_i^t \vee s_{i+1}^t)$$

Like in every elementary cellular automaton, there are 2^3 possible neighbourhood configurations, where in this case three are mapped onto state 0 and five onto state 1.

For a single cell with state 1 in the middle of the first line as initial condition, the rule produces the following time-space diagram:



Figure 3.24: Rule 110, 10 time steps

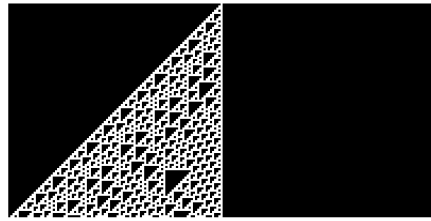


Figure 3.25: Rule 110, 100 time steps

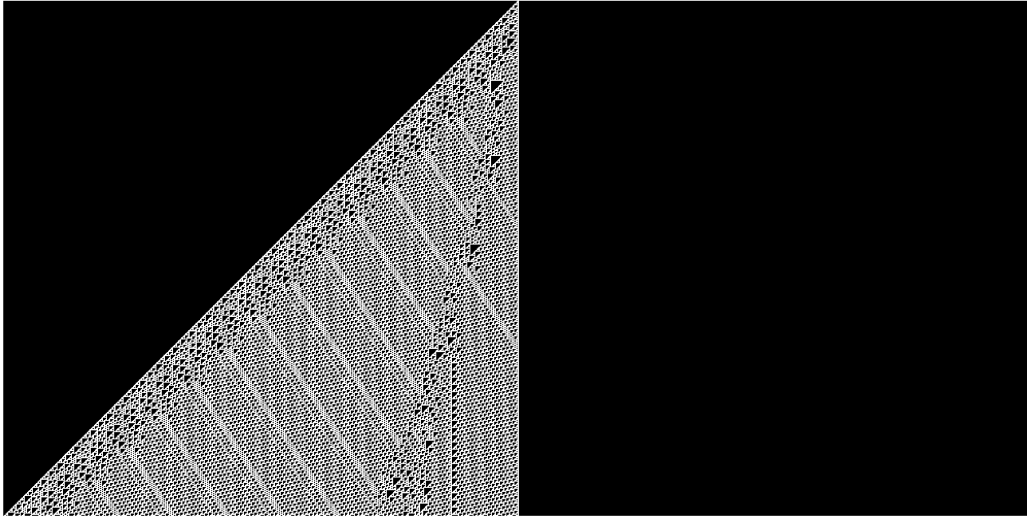


Figure 3.26: Rule 110, 500 time steps

As already mentioned, this rule belongs to Class 4. We can observe both areas with regularities that are reminiscent of Elementary Class 2 behaviour, as well as irregular patterns. We can also observe lines of triangles that move in an irregular manner downwards to the bottom.

Cook uses a sophisticated construction to prove the universality of rule 110. As mentioned above, to prove that a system is capable of universal computation, it is necessary to prove that it is capable of emulating a Turing machine. In order to do this, Cook uses several intermediate steps.

Tag Systems

The starting point of his proof are so-called tag systems, which are machines that operate on a finite tape. The machine operates by reading symbols off the front of the tape and writing symbols at the end of the tape. In each time step, two symbols are read at the beginning of the tape which are then removed. Based on the first symbol and a lookup table, the machine decides which symbols to write at the end of the tape

[4].

Example: Consider the tape ACDABBE. Suppose the lookup table defines CCDD for the symbol A. The machine reads the beginning of the tape and finds A. It will remove the first two symbols AC and append CCDD to the end, resulting in DABBECCD [4].

It is possible to show that tag systems are capable of emulating a Turing machine hence are capable of universal computation.

Cyclic Tag Systems

Next, the author considers so-called cyclic tag systems, which are similar to tag systems but do not need a lookup table to produce output. As it is the case with tag systems, these machines operate on a finite tape and are read from the front. But in contrast to tag systems, cyclic tag systems use only an array of appendants. For each read operation, the next item in the appendants list is appended to the end of the tape if a specified condition is met by the current read operation. The machine decides then, whether it will append the current value of the appendants to the list or not. If the end of the appendants list is reached, the machine goes back to the beginning of the list. Cyclic tag systems only use a two letter alphabet, like $\{0, 1\}$. If the machine reads 1 it will append the current appendant to the list, if it reads 0 it will not append it.

It is possible to show that cyclic tag systems can emulate tag systems and therefore are capable of universal computation.

Glider System

The last class of machines used for Cook's proof are glider systems. The alphabet of these machines consists of so-called gliders, which are particles that move through space with a predefined direction and speed. In the one dimensional case, the space is just a line on which the gliders can move. Events are triggered, when two gliders meet.

A lookup table defines the result of this collision. This could be, for example, a set of new gliders that move away from the collision point.

Rule 110

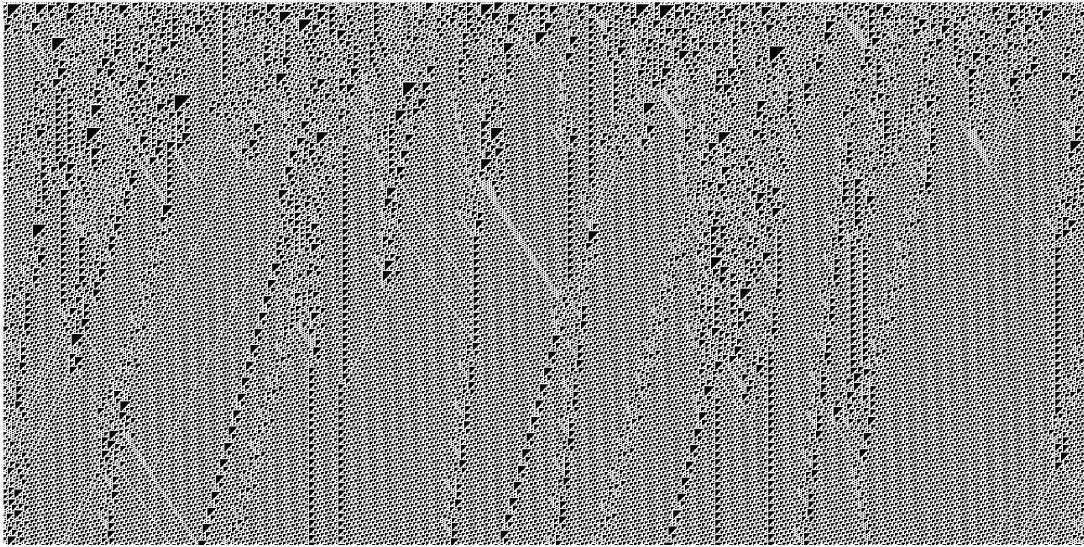


Figure 3.27: Rule 110, 500 time steps, random initial condition

The main idea of Cook's proof is to construct a glider system which in turn can be used to simulate a cyclic tag system. The same approach is used to prove universality of rule 110. The author identifies gliders in the time-space evolution of rule 110 which enables him to construct a cyclic tag system with rule 110.

Figure 3.27 shows the time-space evolution for 500 time steps of rule 110 from random initial conditions. As we can see visually, the behaviour of the automaton is not completely random. Rather we can identify lines of bigger triangles moving through time-space on a background consisting of little triangles. The bigger triangles are the gliders and the background is called the *ether*.

Cook identified 8 different types of gliders in rule 110 and a structure that produces gliders, which he called a *glider gun*. All types of gliders also have subtypes. The

different types and their interaction is analysed and the author identifies certain types used for his construction of a cyclic tag system with rule 110. The whole proof with all details of the construction can be found in [4].

3.8 Automata with Extended Neighbourhood and State Set

Several generalizations can be made within one dimensional cellular automata. In the above discussions, we only considered the most simple case, where $\text{card } \mathcal{S} = 2$ and the neighbourhood radius $r = 1$. Obviously, it is easily possible to extend the state set as well as the neighbourhood radius. Let $k = \text{card } \mathcal{S}$ (the cardinality of the state set) and r the radius of the neighbourhood, so that the state of cell i in time step $t + 1$ is computed as follows:

$$s_i^{t+1} = \mathcal{F}(s_{i-r}^t, \dots, s_{i+r}^t)$$

Here we can identify k^{2r+1} possible local rules [41]. This immense quantity, even for small k and s makes empirical analysis very difficult. WOLFRAM discovered, however, that also automata with extended radius and state set fall into the above mentioned four categories [41].

Also in this more general case, many properties of such automata only depend on the class in which they lie and are independent of both the initial configuration and the exact rule function. See [41] for a in-depth discussion of this fact. This also applies to generalized Elementary cellular automata with extended state set and radius.

In the following section we consider two examples of one dimensional cellular automata with extended neighbourhood radius. The state set is still $\mathcal{S} = \{0, 1\}$, but we extend the neighbourhood radius r to $r = 2$.

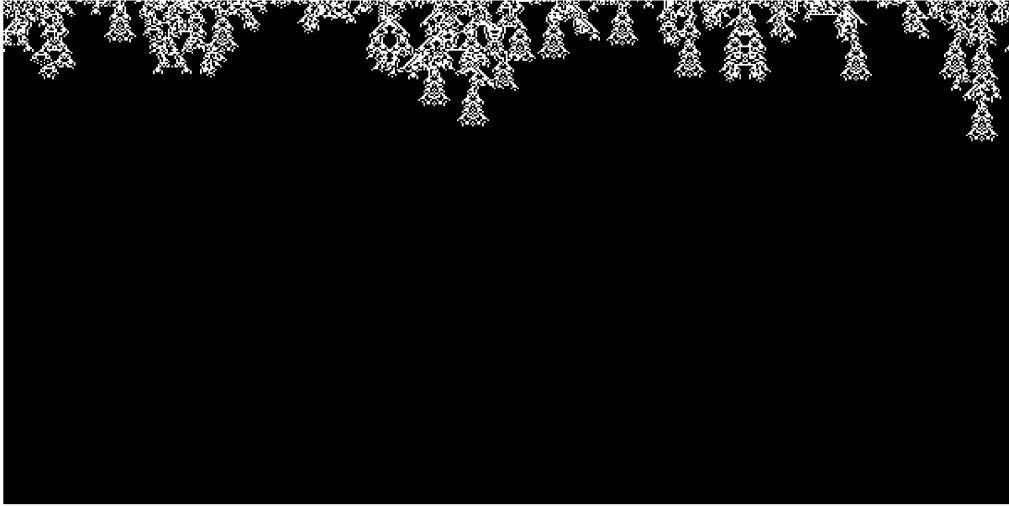


Figure 3.28: 300 time steps

3.8.1 Class 4 Rule with Extended Neighbourhood Radius

This automaton was described by WOLFRAM in [40]. The local update function is defined as follows:

$$s_i^{t+1} = \begin{cases} s_i^t + 1 \mod 2 & \text{if two or four neighbours fulfil } s^t = 1 \\ s_i^t & \text{if one or three neighbours fulfil } s^t = 1 \\ 0 & \text{else} \end{cases}$$

The time-space diagrams in Figure 3.28 and Figure 3.29 are produced by this cellular automaton with random initial conditions in 300 time steps. We can observe that the cellular automaton soon reaches a stable configuration in which all cells have the same state. These images are calculated with a slightly modified version of the script in Appendix 1.

Depending on the initial condition it is also possible that a periodic configuration emerges. A repetitive structure of living cells is the result. An example can be seen in Figure 3.29.

Let us take a closer look into Figure 3.29. After a consolidation phase in the first 50-

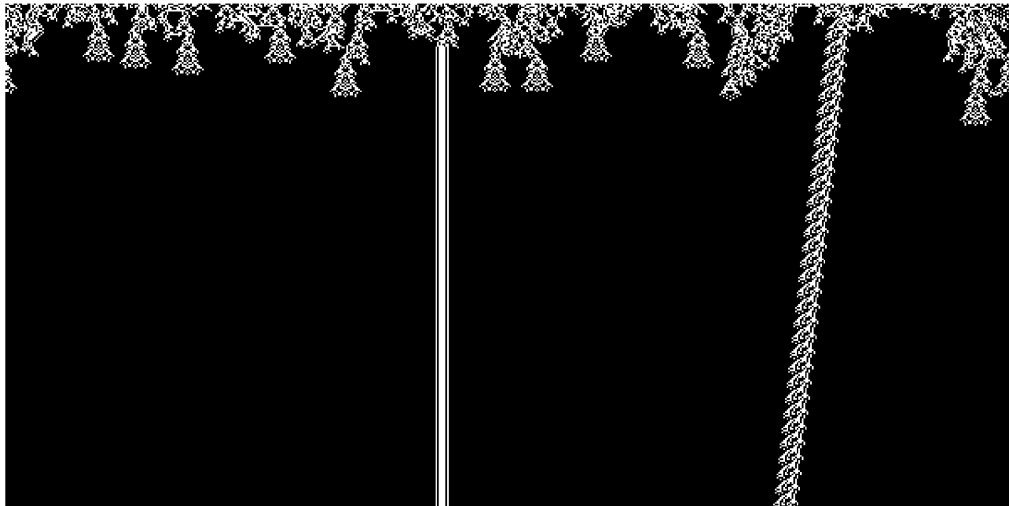


Figure 3.29: 300 time steps

60 time steps, two persistent structures emerge. We can identify the patterns which generate these stable structures and ask what happens, when the two stable structures meet later on. The structure on the left-hand side, which moves straight downwards is generated by:

```
001111110101
```

And the stable structure on the right-hand side, which moves slightly diagonal from right to left is generated by the pattern:

```
1000111111110101101
```

If these structures were alone in their world, they could both live on forever. Given a sufficient distance (≥ 3) between each other, many of them could survive in one universe until infinity without ever affecting each other. This can be seen in Figures 3.30 and 3.31.

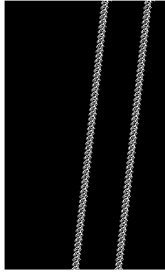


Figure 3.30: Two stable structures

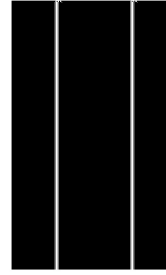


Figure 3.31: Other stable structures

Let us now put both in one world and see in Figures 3.32 and 3.33 what happens when they meet:

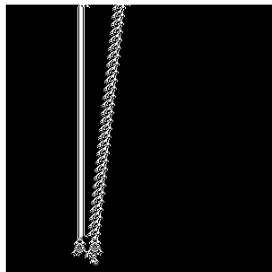


Figure 3.32: Sad meeting

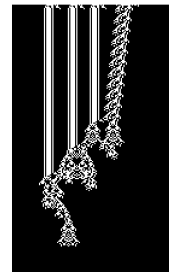


Figure 3.33: Sad meeting 2

As we can see in the time-space diagram, the structures annihilate themselves soon after they encounter each other. Even if we construct a world where two or three "straight-line species" live together, it can happen that they all disappear when they encounter the other "species".

Even though not mathematically relevant it is interesting to think about similarities between life on planet Earth or in the universe and this simple program. Note also, that interaction between structures does not lead to their annihilation automatically. In Figure 3.29 the stable structures actually *emerged* due to some interaction between other structures. The conditions for stable living structures to emerge in this simple world, however, are very difficult to achieve. A slight change in the provided initial

conditions may lead to a completely different evolution of the system. For almost all initial conditions, no persisting structures would emerge at all. This reminds one of the conditions necessary for life to emerge on a planet.

3.8.2 Finite Width Example and Computational Tests

Figures 3.34, 3.35 and 3.36 show an example of a one dimensional cellular automaton with extended ($r = 2$) neighbourhood and finite width. It uses periodic boundary conditions and computational experiments revealed some interesting properties. The state set is: $\{0, 1\}$ and the automaton has the following transition function:

$$s_i^{t+1} = s_{i-2}^t + s_{i+1}^t + s_{i-2}^t \cdot s_{i+1}^t + s_{i-2}^t \cdot s_{i-1}^t \cdot s_{i+1}^t \mod 2$$

Figures 3.34 and 3.36 use a single cell in state 1 in the middle of the first row (single-seed) and Figure 3.35 uses random initial conditions. The only difference between Figure 3.34 and 3.36 is that in the latter we added one more column, so we obtain a 300×301 matrix. The universe is then 301 cells wide. This single additional column changes a lot in the behaviour of the system because it leads to a slightly different "overlapping" caused by the periodic boundary conditions.

The result is interesting. If one interaction takes place, a slightly distorted view of the the Sierpinski triangle emerges, which occurs often in elementary cellular automata as we have seen. If the interaction happens three or more times, we obtain deterministic chaos, as seen in the lower edges of Figure 3.34. Whether or not the overlapping leads to a certain behaviour depends on the width of the matrix. Let $n \in \mathbb{N}$, then the $m \times n$ matrix generates a regular structure as seen in Figure 3.34, if

$$n \mod 3 = 0.$$

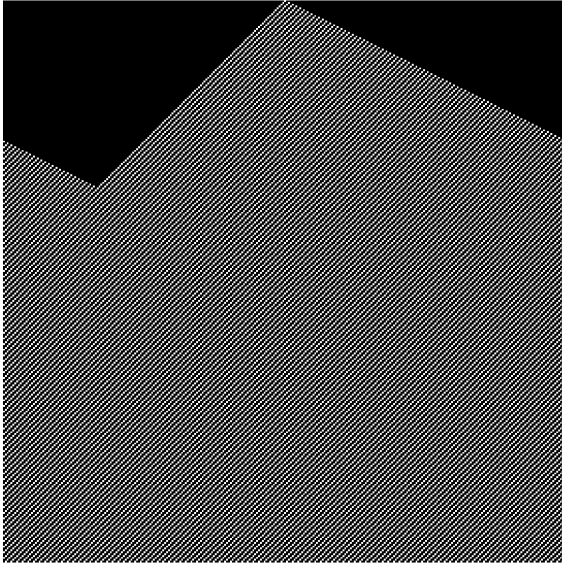


Figure 3.34: 300×300

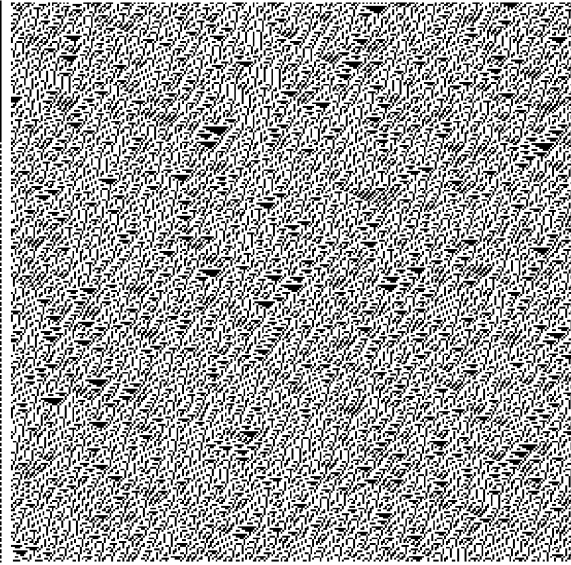


Figure 3.35: 300×300 , random

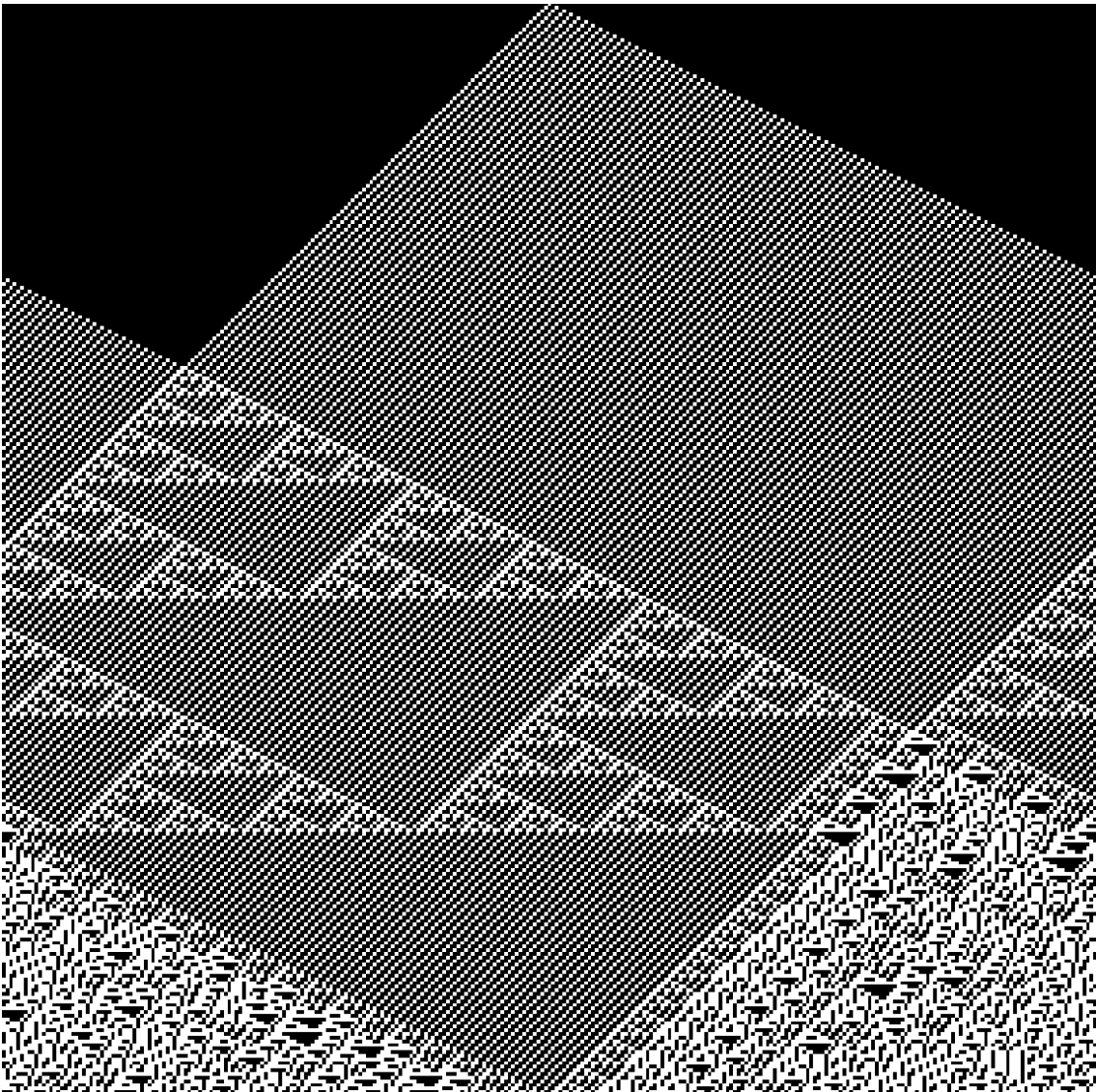


Figure 3.36: 300×301

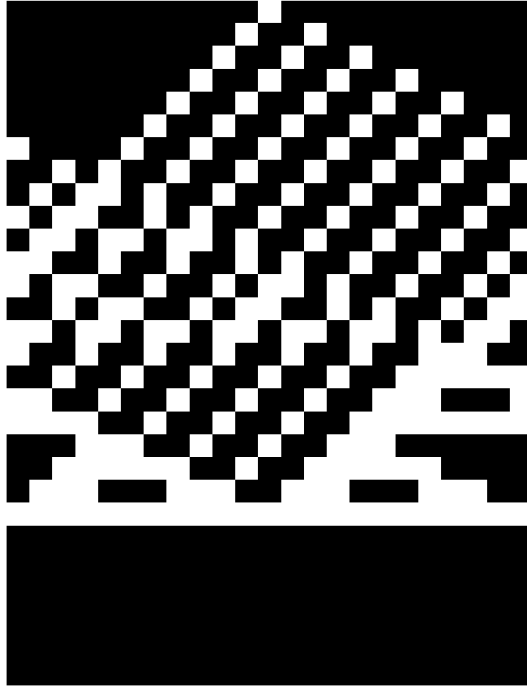


Figure 3.37: 30×23

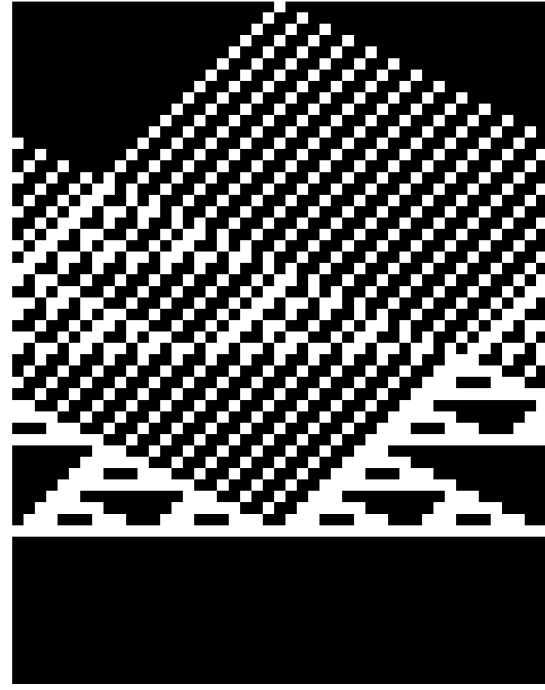


Figure 3.38: 60×47

If however $n \bmod 3 \neq 0$, the above mentioned interference occurs. This is easy to understand since, if we would use an infinite universe, the evolution of each cell would be periodic with a period of length 3.

Looking at the transition function, we see that 0 is a quiescent state. Once all cells are in this state, the configuration does not change any more. Whether or not such a configuration is reached from a single-seed starting condition only depends on n , the width of the universe.

One might be interested for which widths n this is the case. We systematically tested for $n \in [5, 2000]$ and found that this situation occurs for:

$$n \in \{5, 10, 11, 23, 47, 95, 191, 383, 767, 1535\}.$$

For $i > 10$, these numbers are so called *Thâbit numbers*, named after the 9th Century mathematician THĀBIT IBN QURRA. The i -th Thâbit number has the closed form:

$$(n)_i = 3 \cdot 2^i - 1.$$

The next numbers in this sequence are 3071 and 6143, which were in turn tested as well. Also in these cases, the behaviour was as expected. It is noteworthy that the quiescent state is reached in a universe with width n after exactly n time steps.

The image in Figure 3.39 shows the example for a row with 767 cells, for which 780 time steps were calculated. At the bottom it is visible that all cells are in state 0. Interestingly, this calculation reveals that if the matrix-width is a Thâbit number, the distorted view of the the Sierpinski triangle occurs again later, but with a different "background". In contrast to Figure 3.34 where 3 "overlappings" led to deterministic chaos, they produce a highly regular structure in this case, as seen in the lower edges of Figure 3.39.

Since the automaton is deterministic and finite, its maximum cycle length is 2^n . Therefore also for all $n \in \mathbb{N}$ which satisfy $n \bmod 3 \neq 0$ and $\nexists i \in \mathbb{N} : n = 3 \cdot 2^i - 1$, there has to be some $k \in \mathbb{N}$ such that the automaton is periodic with a period of length $k \leq 2^n$. Tables 3.1 and 3.2 show the computational results for the period-lengths for $n \leq 35$ that emerge from single-seed starting conditions. Time-space diagrams for $n \in \{4, 7, 8, 17\}$ are shown in Figures 3.40, 3.41, 3.42 and 3.43.

An additional detail is revealed by analysing the formula for these numbers. Let $f(x) = 3 \cdot 2^x - 1$. Looking (just for fun) for the real root, we find

$$f(x) = 0 \Rightarrow x = -\frac{\log 3}{\log 2}$$

whose absolute value is the fractal dimension of the Sierpinski triangle.

The algorithms developed for both the identification of the quiescent state and the calculation of the period-length are very simple. The first calculates the matrix for a given number of time steps and then checks the row sums to identify a row whose sum is 0. In a second step, the program fixes one (non-zero) row and compares this configuration to the other rows backwards in time. If a match occurs, the program returns the difference of the two respective time steps. The further evolution is known once this period-length is detected.

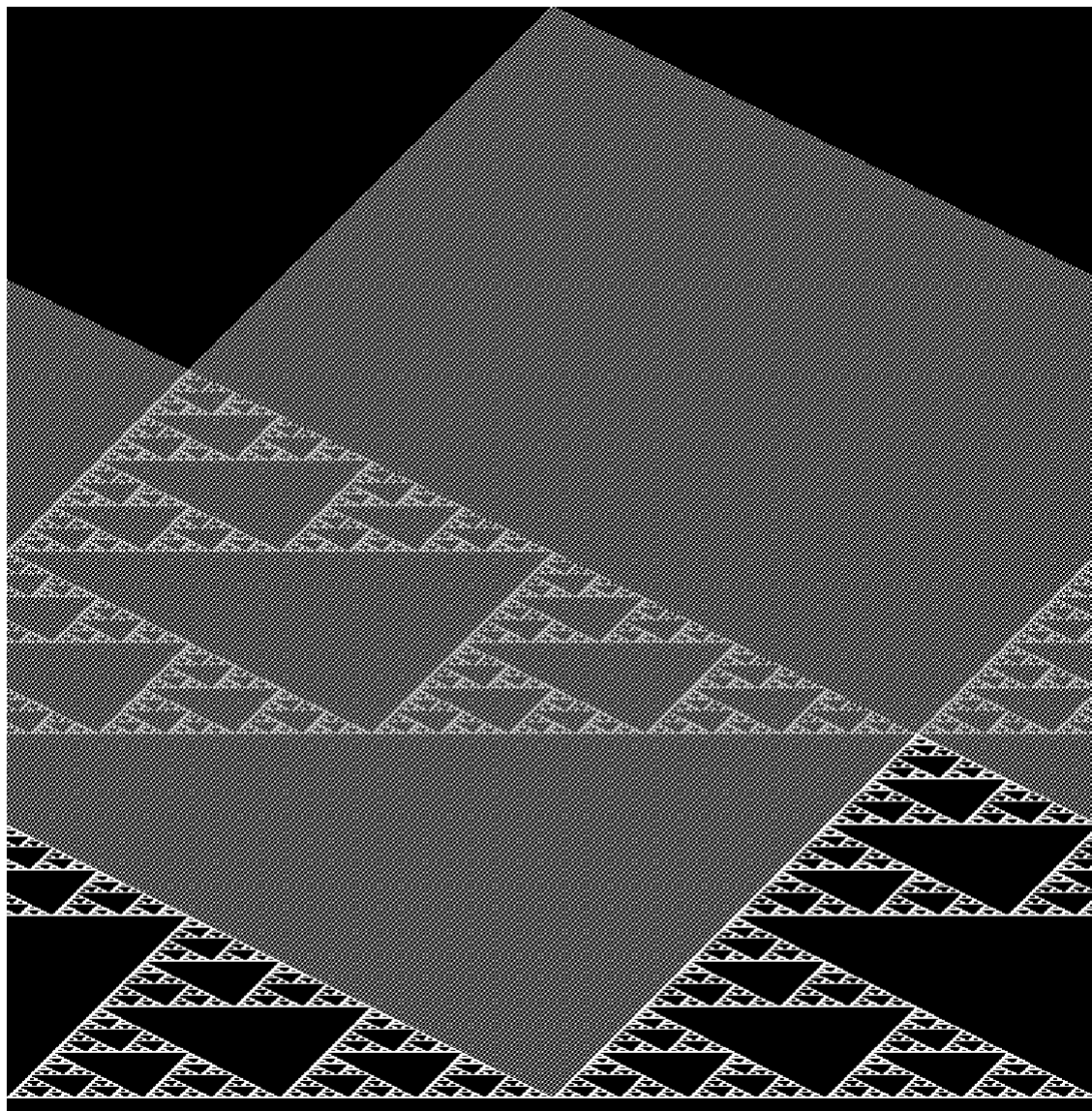


Figure 3.39: $n = 767$

n	Period length
4	4
7	7
8	1
13	130
14	44
16	960
17	34
19	76
20	336

Table 3.1: Period length

n	Period length
22	4136
25	5450
26	2600
28	1992
29	5307
31	3627
32	917
34	27234
35	18690

Table 3.2: Period length 2

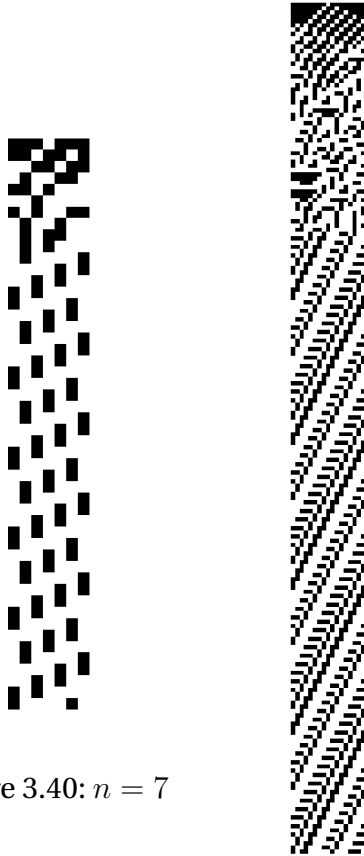


Figure 3.40: $n = 7$



Figure 3.41: $n = 17$

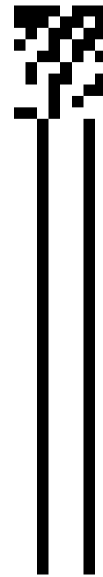


Figure 3.42: $n = 8$



Figure 3.43: $n = 4$

CHAPTER 4 Two Dimensional Cellular Automata

Until this point, we only considered one dimensional cellular automata. Their universe was $\mathbb{Z}^1$ and therefore everything happened on a single line of cells. The two dimensional representations that we have seen were time-space diagrams of the evolution of the systems. This ability to provide a two dimensional image of the time-space evolution of such automata might be an explanation why this simple class of cellular automata gained so much attention, also within the non-scientific community.

In the case of two dimensional cellular automata, the visual representation of the system is more complicated. There are two possibilities: Either we represent the system in each time step next to each other, which is only possible for a very limited amount of time steps, or we delete the representation of the system at time step t once the configuration in $t + 1$ is calculated and continuously update the grid with the new configuration at a certain frame rate. The second approach offers a convenient method of analysing the long-term behaviour of a given starting configuration, while the first approach is useful to conduct an in-depth analysis of the behaviour of small scale patterns over a limited number of time steps.

The only difference to the one dimensional case is that for two dimensional cellular automata the underlying topology is $\mathbb{Z}^2$, hence an infinite chequerboard of cells. The general definition provided in the first chapter is applicable to the two dimensional case as well.

4.1 Overview

In Elementary cellular automata, we considered a state set with $|\mathcal{S}| = 2$ and our neighbourhood consisted of only the left and right neighbours. We saw in the discussion of cellular automata with extended neighbourhood and state set, that even a small increase in these parameters led to a massive increase of possible transition functions. In the two dimensional case this increase is even stronger.

As mentioned in the introduction, two neighbourhoods are the most important in $\mathbb{Z}^2$: the *Moore-Neighbourhood* and the *Von-Neumann-Neighbourhood*. If $|\mathcal{S}| = 2$ and if we consider a Moore-Neighbourhood with radius $r = 1$, we already obtain $2^9 = 512$ possible neighbourhood configurations. We can now ask, how many transition functions we can find, that map the state of the middle cell to its next state. As there are 512 possible neighbourhood configurations and two states, we find 2^{512} different transition functions. Even for a state set with cardinality 2 the number of possible rule functions already exceeds the number of atoms in the universe.

The most famous example of a transition function for a two dimensional cellular automaton is the so-called *Game of Life* which was defined by JOHN CONWAY in 1970. The inventor himself was obviously fascinated by his discovery and stated:

It is probable, given a large enough 'Life' space, initially in a random state, that after a long time, intelligent self-reproducing animals will emerge and populate some parts of the space.

The technological revolution and the increased availability of personal computers in the second half of the 20th century made it possible to dynamically simulate the time-space evolution of cellular automata.

Since the Game of Life became so famous and has a worldwide fan base both within

the general public and the scientific community, we will now discuss this example in more detail.

4.2 Game of Life

There is a huge amount of literature on the Game of Life. Our discussion is mainly based on [25] and [11].

Definition 25. Game of Life: Let $\mathcal{A} = (\mathcal{S}, \mathcal{N}, \mathcal{F}, \mathcal{T})$ where the state set $\mathcal{S} = \{0, 1\}$, the underlying topology $\mathcal{T} = \mathbb{Z}^2$ and the neighbourhood $\mathcal{N}$ is the Moore-Neighbourhood. Let (i, j) be a cell referred to by its Cartesian coordinates on $\mathbb{Z}^2$ and let s_{ij}^t be the state of cell (i, j) at time step t . Let $\sum_{moore} s_{kl}^t$ be the sum of the living cells in the Moore-Neighbourhood of a given cell including that cell as well. Then we can give the following compact definition of the local update rule $\mathcal{F}$ of the Game of Life:

$$s_{ij}^{t+1} = \begin{cases} 1 & \text{if } \sum_{moore} s_{kl}^t = 3 \\ 1 & \text{if } s_{ij}^t = 1 \text{ and } \sum_{moore} s_{kl}^t = 4 \\ 0 & \text{else} \end{cases}$$

The cell states are referred to as *dead* for the state of 0 and *alive* for the state of 1. Usually dead cells are drawn white and living cells black. The colouring of the entire grid at a certain time step represents the configuration of the Game of Life. The local neighbourhood of a cell is the Moore-Neighbourhood and the state of the cells in the local neighbourhood at a given time step determine the state of the cell in the next time step. The rule function as defined above generates the following behaviour:

- If a cell is alive, it stays alive if and only if there are exactly two or three living cells in the neighbourhood. If there is only one living cell in the neighbourhood, the cell dies due to isolation and if there are more than three living neighbours, the cell dies due to overcrowding.

- If a cell is dead, it may become alive if it has exactly three living neighbours. If we think about the cell as a living organism, it has three parents in the Game of Life.

Figure 4.1 shows five time steps in the Game of Life:

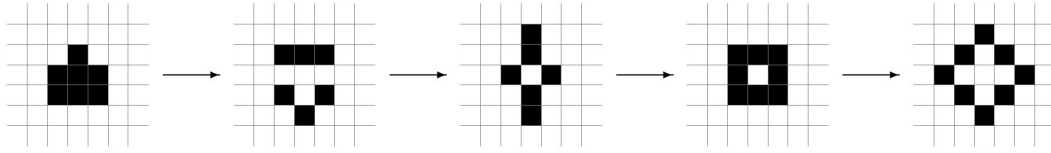


Figure 4.1: Five steps in the Game of Life[11]

As usual, it is necessary to provide the system with some starting configuration, that is, determine which cells are alive and which are dead at $t = 0$. The evolution then follows the above given simple set of rules and even though this evolution is strictly deterministic, it is often surprising and highly complex. When starting the Game with a random initial configuration c_0 , two situations occur quite frequently: Depending on the orbit of c_0 , we often see **eventually fixed** or **eventually temporally periodic** configurations as defined above. We can also observe the birth and death of bigger groups of cells. Famous patterns include the following:

Still-life-configurations: A fixed point pattern, where the global update rule does not change the state of any cell. The most simple configuration which shows this behaviour is of course a configuration with no life at all. The simplest non-empty configuration is the so-called *block* which is a square with a side length of two cells. Game of Life enthusiasts identified and named many other non-trivial still life configurations over the years. Figure 4.2 shows some famous examples of still life configurations.

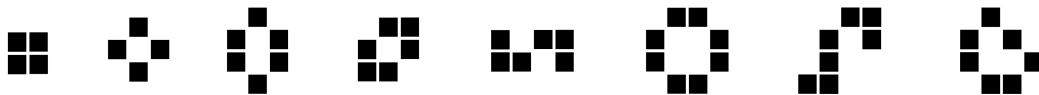


Figure 4.2: Some common invariant forms (l to r): block, tub, beehive, ship, snake, pond, fishhook or eater, loaf [25]

Oscillator-configurations: A temporally periodic pattern, where the global rule changes the state of cells, but a configuration c_0 is reached again every n time steps for a given $n \in \mathbb{N}$. Figure 4.3 shows some examples for $n = 2$.

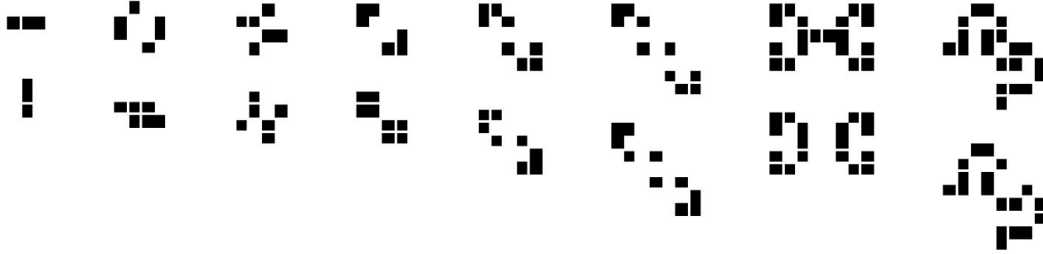


Figure 4.3: Some period 2 oscillators[25]

There are also more complex oscillator configurations. One is known as the CP-Pulsar and is shown in Figure 4.4. The image represents its three different states. After the third iteration, the structure returns to the first configuration again.

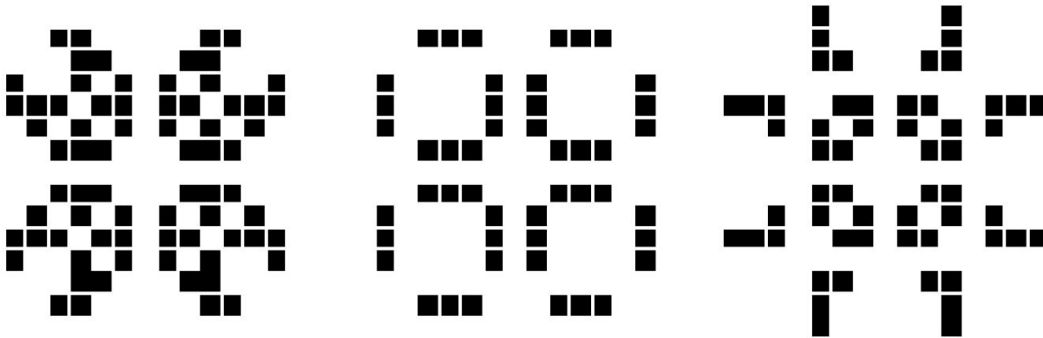


Figure 4.4: The CP-Pulsar[25]

Spaceships: An oscillator-configuration, where the same constellation of living cells appears at an other position on the grid after some time steps. The most famous example for a spaceship configuration is the so-called *glider*:

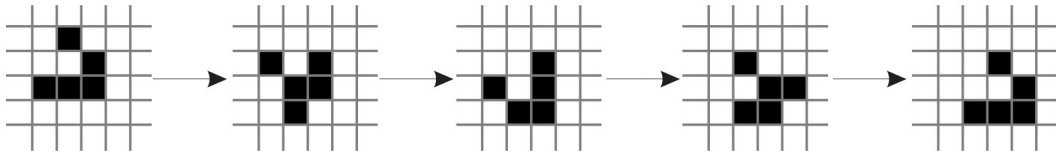


Figure 4.5: The glider[11]

Guns: These are similar to oscillator-configurations because they return to their initial configuration after a certain number of time steps but additionally they emit spaceships periodically.

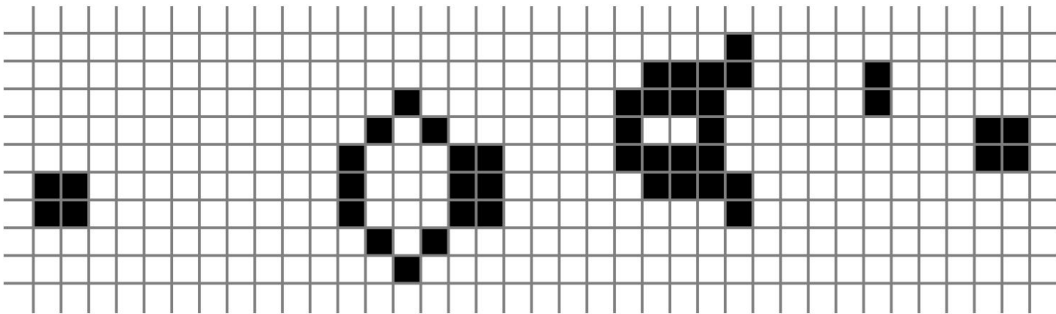


Figure 4.6: A glider gun[11]

These were one of the most remarkable discoveries in the Game of Life, since CONWAY conjectured, that the number of living cells cannot grow without bound in the Game of Life. The existence of a glider gun, elegantly disproved this conjecture. By that time, a special arrangement of 13 gliders was found that produce a glider gun. Since this time, many such configurations have been found. Also a gun that produces a glider only every 256 time steps was found. Since these structures can be seen as a clock in the Game of Life, they also play a major role in the proof of universality for this system.

In the Game of Life, information can theoretically flow with a maximum speed of one cell per iteration. This is called the *speed of light*, since in the real world the speed of light acts as the upper bound for the velocity with which information can propagate through time-space. It was proven by CONWAY, that the maximum speed with which

objects can actually move in Game of Life is $\frac{c}{2}$. He invented the name *spaceships* for objects that actually move on the grid with this speed. The above shown glider moves with $\frac{c}{4}$.

When the Game of Life became known, it attracted much attention also in the non-scientific community. Many people were fascinated about how such a simple construction is able to produce such complex behaviour. We will now examine some basic mathematical properties of the Game of Life based on the theory described above.

4.2.1 Mathematical Properties of the Transition Function

Proposition 7. The Game of Life is not surjective.

Proof: Consider a configuration c_1 , where all cells are in state 0 and another configuration c_2 , where all cells are in state 0 except of one, which is in state 1. Both configurations have the same image under the transition function. The restricted function G_F is hence not injective. It follows directly from the Moore-Myhill Theorem that the Game of Life is not surjective.

□

Corollary 4. The Game of Life is not reversible.

Proof: Since the transition function is not surjective, it is not bijective and hence there is no inverse function.

□

It was already known since its invention, that the Game of Life is not surjective. As we have seen, this implies that there must be Garden-of-Eden configurations and orphans.

The search for such configurations is however very difficult. The first orphan was discovered already in 1971 by ROGER BANKS and consisted of 297 cells aligned in a

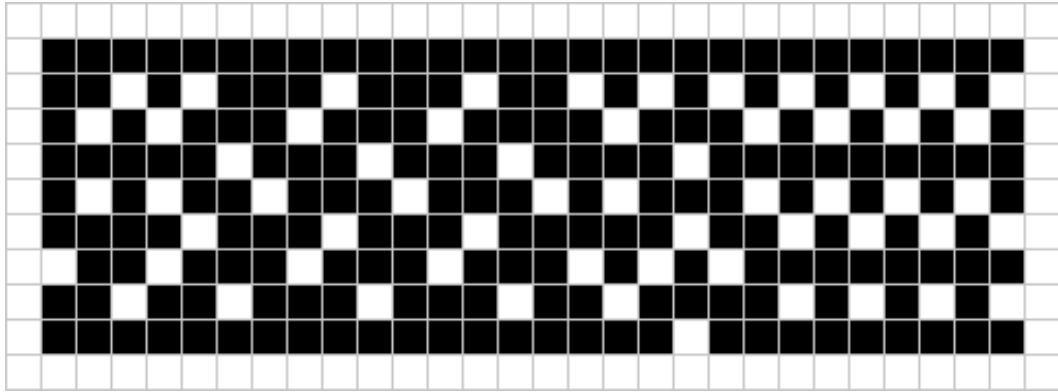


Figure 4.7: First orphan¹

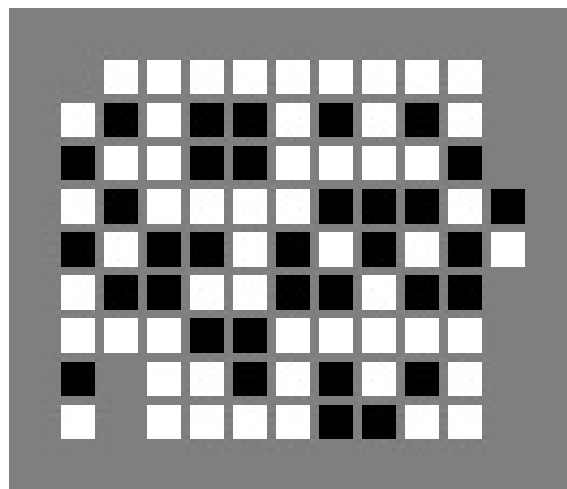


Figure 4.8: Current record²

rectangle of height 9 and width 33. 226 cells where in state 1. In Figure 4.7, we see this famous discovery.

Since 1971, many scientists searched for smaller orphans in the Game of Life. Four researchers made significant contributions to this task. ACHIM FLAMMENKAMP, NICOLAY BELUCHENKO, MARIJN HEULE and STEVEN EKER. In 1991 FLAMMENKAMP discovered an orphan which was much smaller than the first orphan discovered by BANKS. The orphan was only 14x14 in size, hence consisting of only 196 cells. In 2011, MARIJN HEULE

¹Source: http://www.conwaylife.com/wiki/File:Gardenofeden1_cropped.png

²Source: http://www.homes.uni-bielefeld.de/achim/orphan_10th.html

ET. AL. searched the whole space of 10x10 patterns and they could show that their cannot exist Garden-of-Eden configurations smaller than 6x6. They also found a 10x10 Garden-of-Eden configuration, which was the smallest at that time. On December 16th in 2016 however, EKER found a 9x11 configuration, which is the smallest orphan currently known. Figure 4.8 shows an image of this recent discovery. The search for smaller orphans continues and it is likely that soon there will be another record.

4.3 Langton's Vant

Another famous example of a two dimensional cellular automaton was developed by LANGTON. In [13] he asked the following question:

If living organisms are composed of molecules that are intrinsically inanimate, why is it that living matter differs so radically from nonliving matter, which also consists of inanimate molecules? Why does the living organism appear to be more than the sum of its inanimate parts?

In this question he actually addresses the idea of emergent phenomena, which are often mentioned in the context of cellular automata. Emergent phenomena can be described as outcomes of deterministic interactions that are somewhat more than the parts they consist of. In nature, we often encounter such phenomena. Life by itself or self-consciousness are often mentioned in this context. Also the term *society* refers to an emergent phenomenon as a complex outcome of mainly localized interactions between individuals. Also in the realm of discrete dynamical systems like cellular automata we encounter such behaviour, as we have already seen in previous chapters.

LANGTON wanted to simulate the "molecular logic of the living state" with an approach he called artificial biochemistry. To develop a computational model for his approach, he used cellular automata. He first describes the properties he expects from a compu-

tational model to suit his needs and finds that cellular automata are the best match. Since his research took place in 1986, he could already build upon some works of WOLFRAM within this field.

He states that colonies of social insects provide a good basis to study artificial life, as the local interactions of very simple individuals exhibit enormous complexity as a whole. He then outlines the idea of *virtual ants* which he called *Vants*.

The model is very simple. A Vant is a special cell on a $\mathbb{Z}^2$ grid where each cell (i, j) can take one of two states $s \in \mathcal{S}$ where $\mathcal{S} = \{\text{to-right}, \text{to-left}\}$. The Vant itself is often drawn as a V-shaped object that, in each time step, moves one cell on the grid according to some basic rules:

- On a cell in state `to-right` turn 90° right, change the state of the cell to `to-left` and move one step forward.
- On a cell in state `to-left` turn 90° left, change the state of the cell to `to-right` and move one step forward.

The obvious difference to the systems we have seen so far, is that we have something like a 'moving object'. In fact this object can be modelled as a normal cell when we extend the state set of the system to 8 different states. Each state encodes the four possible looking directions of the ant combined with the two states `to-left` and `to-right`. For each cell we can determine its state in time step $t + 1$ by looking at the cells in the Von-Neumann-Neighbourhood at time step t . The state of the cell under consideration will change if and only if the Vant will reach this cell in the next time step.

The model produces remarkable complexity, which one might not expect given the simple rules. For the simulation we used the Software Netlogo 6.0.1 [38] and slightly adapted a model developed by WILENSKY in [37]. The adaptations are just for enhanc-

ing the quality of the images presented here. We use a 240×240 grid for the simulation. A white cell is in state `to-right`, and a black cell is in state `to-left`.

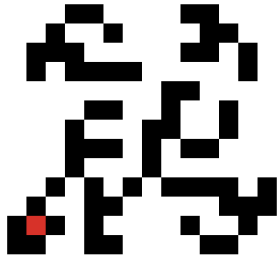


Figure 4.9: After 500 iterations



Figure 4.10: After 3000 iterations

As we can see in Figures 4.9 and 4.9, in the first iterations nothing special happens. The Vant wanders around its starting point and traces a pseudo-random path. Even after 10000 iterations, nothing special appears to happen. But at iteration 10360, we observe, that the automaton enters a state, in which the Vant starts to build a straight path along the grid, which she does not leave any more. This behaviour is reminiscent of the gliders observed in the Game of Life.



Figure 4.11: After 10000 iterations



Figure 4.12: After 10360 iterations

The automaton has produced some kind of order out of chaos, a property of cellular automata that we have already encountered numerous times throughout this thesis. Figure 4.13 shows this behavior and the structure, which is sometimes called "the highway".

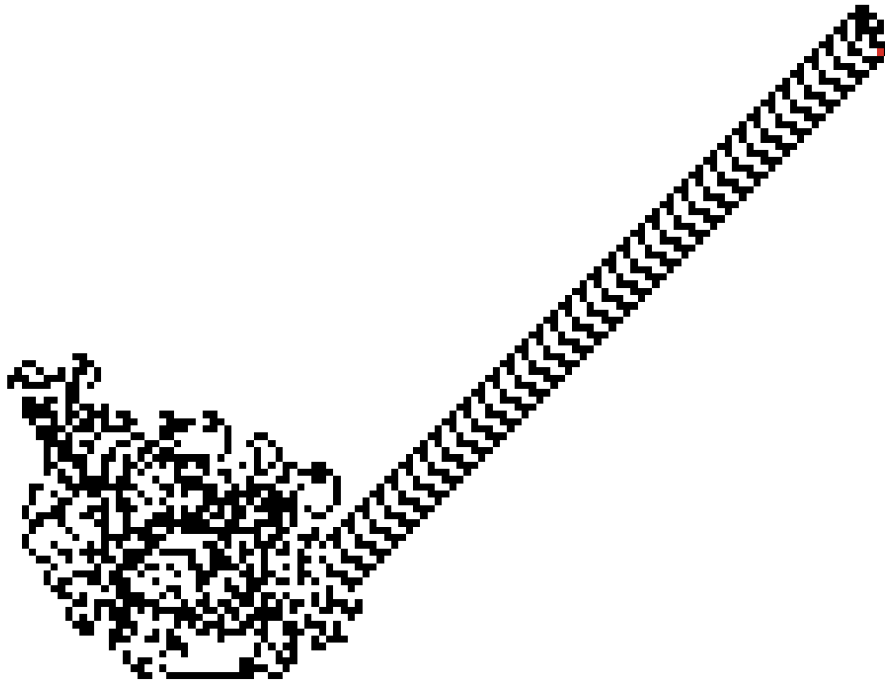


Figure 4.13: After 13842 iterations

4.3.1 Theoretical Properties of Vant

In 2000 GAJARDO, MOREIRA and GLOES could show in [6] that the system is Turing complete. This is the most remarkable theoretical result for Vant. Before this research was published, very little theoretical properties of Vant were known. One was, that for any initial configuration, the trajectory of the ant is unbounded [6]. It could be shown that this means that the set of cells that are visited infinitely often by the ant has no corners, where a corner of a set is a cell, for which the number of neighbours which do not belong to the set is greater or equal than two.

Suppose an s -finite configuration for Vant (as we have defined above this is a configuration with finite s -support which in turn means that the number of cells which are not in state s is finite). It has been conjectured that for any initial condition with finite support the Vant starts building the highway in some unobstructed direction [6].

Suppose we ask a certain question like: Will the Vant ever reach a certain cell? Then such questions are decidable if the above mentioned conjecture is true since from the time the Vant starts building the highway the future evolution of the system can be easily predicted.

In [6] the authors present a construction through which they are able to simulate the evaluation of any computable algorithm on the system. The input is represented as the initial configuration of certain cells. The calculation itself is performed by the trajectory of a single Vant and the computation result is represented again by the state of certain cells. By this approach the authors could show that the above mentioned problem: *Will the Vant ever reach a certain cell?* is P-hard, by showing that it reduces to a problem known to be P-complete namely the calculation of boolean circuits.

Boolean circuits are computational models for the evaluation of boolean functions like AND or NOT. The function AND for example takes two boolean inputs *A* and *B* and returns `true` if and only if both *A* and *B* are `true` and returns `false` otherwise. The NOT function changes the value of the input from `true` to `false` or from `false` to `true`, respectively. Any boolean function can be formulated as a composition of AND and NOT functions.

The construction presented in [6] is quite simple yet remarkable. The Vant visits a cell which is considered as the input. Since in this model every cell which is not the Vant itself can only take two states, we can consider these states as `true` and `false`. Depending on the value of the input cell, the Vant follows different paths. Consider the NOT function for example. The output cell is in state `false` in the initial configuration and the Vant visits the input cell. If the input is `true`, the Vant exits without changing the state of the output cell. If the input is `false`, the Vant takes a different path, changes the value of the output cell to `true` and then exits. This idea is illustrated in Figure 4.14. It is crucial to ensure that the Vant will never return because this might result in a change of the state of the output cell. This is realized by directing the Vant into a *highway seed*, which forces the Vant to start building a highway. This ensures that

it never returns. Figure 4.15 illustrates this construction, where a black cell represents the `to-right` state and a white cell stands for the `to-left` state.

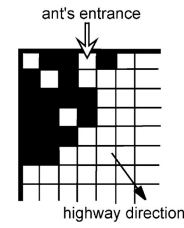
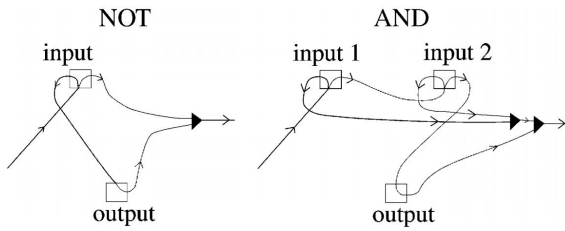


Figure 4.14: Logical gates in Vant[6]

Figure 4.15: Highway constructor[6]

By using this approach the authors could prove the Turing completeness of Langton's Vant. The whole idea and the proof can be found in [6].

4.4 Linear Rules

We already provided a theoretical definition of Linear cellular automata. We will now look at an example and analyse some properties. The following introduction into this special class of two dimensional cellular automata transition functions can be found in [16], [3] and [18]. In [31] and in [32] this approach was also used. At certain points we use slightly different notations and definitions to provide a more formalized definition of the concept.

As mentioned above, even for a state set with cardinality 2 the number of possible rule functions given a Moore-Neighbourhood with radius 1 on $\mathbb{Z}^2$ is 2^{512} , which makes it difficult to systematically analyse the behaviour of these rules due to their large number. One possibility is to only consider some special subset of rules and analyse their behaviour.

In the above mentioned references a classification scheme for a special subset of this rule space is presented. The idea is to enumerate the Moore-Neighbourhood of the

middle cell (2^0) in the following way:

$$\begin{pmatrix} 2^6 & 2^7 & 2^8 \\ 2^5 & 2^0 & 2^1 \\ 2^4 & 2^3 & 2^2 \end{pmatrix}$$

The center cell is the cell under consideration, whose state in the next time step should be calculated according to the states of its Moore-Neighbours. There are nine basic rules: 1, 2, 4, 8, 16, 32, 64, 128 and 256 and each basic rule has its corresponding rule matrix.

Definition 26. Rule Matrix: For a given rule number $k \in [0, 511]$ we calculate a 9 digit binary representation:

$$k = \sum_{i=0}^8 a_i \cdot 2^i$$

and we obtain a sequence of coefficients $(a_i)_{i \in [0,8]}$. We write the coefficients into a matrix in the following way. This matrix we refer to as the **Rule Matrix of rule k** .

$$R_k = \begin{pmatrix} a_6 & a_7 & a_8 \\ a_5 & a_0 & a_1 \\ a_4 & a_3 & a_2 \end{pmatrix}$$

Example:

$$R_{16} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \text{ or } R_{256} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Our rule space consists of 512 rules and each rule matrix R_k , where $k \in [0, 511]$, can be expressed as a composition of basic rules. This composition is obtained by the base 2 representation of the rule number.

Example: Consider R_{170} . We convert 170 to base 2: $170 = 2^7 + 2^5 + 2^3 + 2^1 = 128 + 32 + 8 + 2$. We can obtain the rule matrix for rule 170 by composition of rule 128,

rule 32, rule 8 and rule 2. This is possible using the method described above by writing the coefficients into the rule matrix. This composition of basic rules can simply be defined as a modulo 2 matrix addition of basic rule matrices.

Definition 27. Let $A_{(i,j)}$ and $B_{(i,j)}$ be basic rule matrices.

$$A \oplus B := \sum_{i=0}^2 \sum_{j=0}^2 a_{(i,j)} + b_{(i,j)} \mod 2$$

The rule matrix for rule 170 can hence be obtained by:

$$R_{170} = R_{128} \oplus R_{32} \oplus R_8 \oplus R_2$$

Proposition 8. Let L be the set of the 512 linear rules. For each $l \in L$ there exists a unique composition of basic rules.

Proof: Let $l \in L$. Since we can give a unique base 2 representation of l we can find a unique composition of basic rule matrices.

□

Example: Consider rule 170. Since $170 = 0 \cdot 2^8 + 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$ we obtain the sequence 010101010 and hence the following rule matrix:

$$R_{(i,j)} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Corollary 5. $(L, \oplus)$ is a commutative group.

Proof: All properties are basically inherited from $(\mathbb{Z}_2, +)$ and from matrix addition. Let $m, n, k \in L$.

- $(L, \oplus)$ is algebraically closed since $(\mathbb{Z}_2, +)$ is algebraically closed.

- $(L, \oplus)$ is associative since if $m, n, k \in L$ it is obvious that $(m \oplus n) \oplus k = m \oplus (n \oplus k)$.
- $(L, \oplus)$ is commutative since for $m, n \in L$, $m \oplus n = n \oplus m$.
- Rule 0 is the identity element, since $\forall m \in L : m \oplus 0 = 0 \oplus m = m$. This property is inherited from $(\mathbb{Z}_2, +)$.
- Each $m \in L$ is its own inverse. $\forall m \in L : m \oplus m = 0$, since this is also the case in $(\mathbb{Z}_2, +)$.

□

Definition 28. We define the transition function induced by a given rule matrix $R_{(i,j)}$ as follows: Let $\mathcal{S} = \{0, 1\}$ and $(i, j) \in \mathbb{Z}^2$. Let $s_{(i,j)}^t$ be the state of cell (i, j) at time step t . $s_{(i,j)}^{t+1}$ is obtained by:

$$s_{(i,j)}^{t+1} = \sum_{(i,j): R_{(i,j)} \neq 0} s_{(i,j)}^t \mod 2$$

Within this scheme, rule 1 is the identity function, leaving the configuration invariant under its application.

Example 2: Consider for example rule 170. It declares the following transition function:

$$s_{(i,j)}^{t+1} = s_{(i-1,j)}^t + s_{i,j-1}^t + s_{(i+1,j)}^t + s_{i,j+1}^t \mod 2$$

Let F_{170} be the local transition function for rule 170, then its application to a lattice A induces the following transformation:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \Rightarrow F_{170}(A) = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

Note that in this example all cells which do not lie on the grid are in state 0 which is a matter of definition. For computer simulations of two dimensional cellular automata,

a finite subset of $\mathbb{Z}^2$ is often represented as a torus, to simulate periodic boundary conditions.

The authors in [18] also present the concept of *hybrid rules* or *hybrid cellular automata*. This means that up to every entry in a problem matrix can update its state based on its own transition function. For example, different linear rules can be applied to different rows of a problem matrix.

As we will see in the next chapter, the authors found some applications of these linear rules in digital image processing.

CHAPTER 5 --- Selected Applications of Cellular Automata

In this chapter, we provide an overview of certain applications of cellular automata. First we look at applications of this concept in cryptography. Then we provide some examples on how cellular automata can be applied in digital image processing and finally we describe a famous model used to simulate forest fires, which is based on a two dimensional cellular automaton.

5.1 Applications of Cellular Automata in Cryptography

Some attempts were made to apply cellular automata to problems in cryptography. This section presents a small overview of these approaches.

5.1.1 Random Number Generation

The rise of electronic communication in recent decades has brought an increasing demand on transmission security and hence encryption. One important task within almost every cryptographical system is random number generation. For example, the widespread RSA cryptosystem depends on the generation of large random primes. Also in symmetric cryptosystems it is a crucial task to generate random integers. Such numbers are also needed in the field of stochastic simulation, to simulate the behaviour of a random variable with some given distribution.

To generate random numbers on a computer system, the use of a pseudo-random number generator (PRNG) is widespread. A PRNG is a deterministic algorithm that produces a random sequence of bits based on an initial seed of randomness, like user input or certain hardware properties of the system. The aim is to produce an output that mimics the behaviour of true random sources like radioactive decay.

There are many approaches in PRNG and since this is not the main topic of this thesis, we only refer to some additional sources for further reading on this topic. A useful overview is for example given in [22].

Some attempts have been made to apply Elementary cellular automata as PRNG. This was first done by WOLFRAM in [42]. He presented a stream cipher to encrypt data by using rule 30 as PRNG. Rule 30 describes the following transition function:

$$s_i^{t+1} = s_{i-1}^t + s_i^t + s_{i+1}^t + s_i^t \cdot s_{i+1}^t \mod 2$$

It has the following boolean form:

$$s_i^{t+1} = s_{i-1}^t \vee (s_i^t \vee s_{i+1}^t)$$

From a single seed, the rule produces a time-space diagram as depicted in Figure 5.1.

For a given cell i and a compact interval $T \subseteq \mathbb{N}$ and $t \in T$ the sequence $(s_i)_{t \in T}$ produces a random bit sequence of length $|T|$. Let $(p_t)_{t \in T}$ with $p_t \in \{0, 1\} \forall t \in T$ be a binary plain text, then a cipher text $(c_t)_{t \in T}$ can be produced from the plain text by

$$c_t = p_t \vee s_t.$$

If the sequence (s_t) is known, then p_t can be obtained from c_t .

In [42] WOLFRAM proposes to use the random binary sequence obtained by the values of the states of a given cell in its time-space evolution as a secret for a stream cipher.

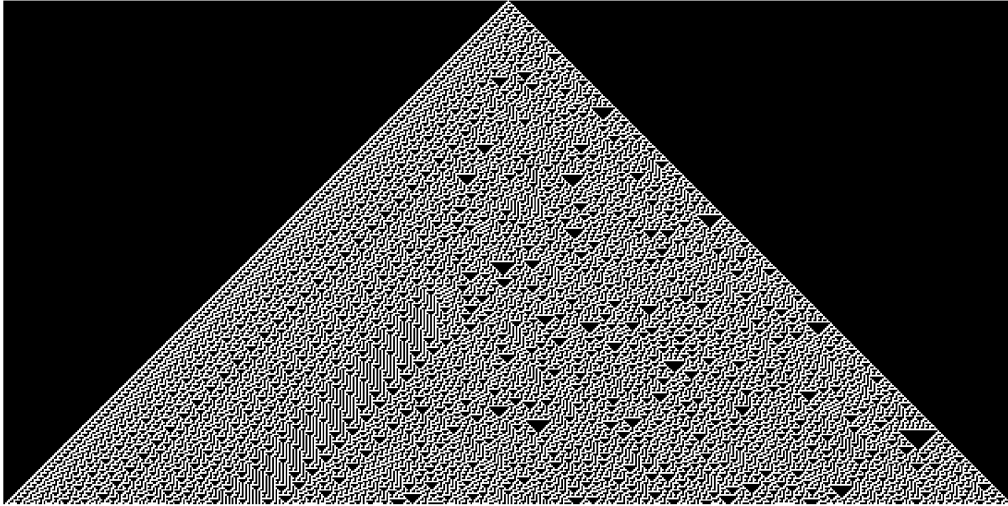


Figure 5.1: Rule 30, 300 time steps

Example: Alice wants to transmit the sequence 1011 to Bob. Let the secret (s_t) be 1001 which is assumed to be known by both parties. Instead of insecurely transmitting the plain text, Alice uses the secret to produce a cipher by the above given instructions. She obtains 0010 which she transmits to Bob who in turn uses the same secret to access the plain text in its original form.

The security of such a cryptographic approach derives from the fact that the secret is hard to guess. Using the evolution of the center cell, no statistical regularities could be found by WOLFRAM which made him assume that this would be a feasible source for random bit sequence generation.

The cryptosystem proposed in [42] was successfully attacked by MEIER and STAFFELBACH in 1991. The details of their attack can be found in [15]. Also the initial claim of WOLFRAM in [42], that recovering the secret was NP-hard, could be rejected by KOC ET AL. Their results are presented in [12].

However, according to the official reference and as described in [34], rule 30 can still be selected as a method for random integer generation in the WOLFRAM LANGUAGE. In [34] some other interesting properties of rule 30 are discussed.

5.1.2 An Asymmetric Encryption Mechanism using Reversible Cellular Automata

Some other ideas of employing cellular automata for cryptography have been proposed as well.

In [7] KARI showed that it is algorithmically undecidable whether a given two or higher dimensional cellular automaton is reversible or not. In the same paper, he showed that given a reversible cellular automaton, it is very hard to compute the corresponding inverse automaton. *Very hard* means that there is no algorithm for finding the inverse rule with a time complexity bounded by a computable function [7].

This fact can be used to define an asymmetric encryption mechanism: The plain text message can be described as a configuration of a reversible cellular automaton $\mathcal{A}$ at a given time step. The encryption is simply done by applying the global transition function for a certain number of time steps. Since $\mathcal{A}$ is reversible, $\mathcal{A}^{-1}$ can be used to obtain the original message again. KARI notes that this can be done very efficiently in parallel if proper hardware implementations are available [7].

Similar to the RSA approach, this method uses a public key which is the automaton $\mathcal{A}$ and a private key, which is its inverse automaton $\mathcal{A}^{-1}$. The public key can be publicly available and is used to encrypt a message *for the owner* of this specific key-pair. Such an approach is known as asymmetric encryption which also solves the key-exchange-problem that arises in symmetric encryption approaches.

There are a number of other approaches on how cellular automata can be employed in cryptography. A good overview is presented in [24].

5.2 Simple Applications of Linear Rules in Image Processing

In [16], [3], [18] and [32] the authors experimented with some applications of linear rules. It could be shown that some of the 512 rules described above can be used for certain purposes in digital image processing.

5.2.1 Translation

In this section, we refer to a digital binary image with $m \times n$ pixels as an $m \times n$ binary matrix.

Consider the basic rules again. It is easy to see that each basic rule induces a translation of images in certain directions.

Example 1: Let F_2 be the local transition function for rule 2, then its application to a given lattice A induces the following translation:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow F_2(A) = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

We can see that every pixel was moved one pixel to the left. Table 5.1 shows the translations induced by the different basic rules. These properties of the basic linear rule functions are obvious if the respective rule matrices are analysed. Consider F_2 , the transition function of rule 2, one more time.

$$F_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

If this function is applied to a given binary image, it basically instructs each pixel to look at his right neighbour. If this neighbour is in state 1 it becomes state 1 in the next

Rule	Translates
R_2	Left
R_{32}	Right
R_8	Top
R_{128}	Bottom
R_4	Diagonal Top-Left
R_{16}	Diagonal Top-Right
R_{64}	Diagonal Bottom-Right
R_{256}	Diagonal Bottom-Left

Table 5.1: Image translation

time step. If the neighbour has state 0 it becomes a 0 pixel in the next time step. So every pixel is moved one pixel to the left. The same logic applies to the other basic transition functions.

5.2.2 Other Basic Image Manipulations and Sweeper's Algorithm

In [3] and [18] the authors explored some other basic image manipulation possibilities with linear rules and hybrid rules. They showed that it is possible to generate multiple copies of an image by applying certain rules. Additionally, they found methods for zooming in, zooming out as well as for thickening and thinning of symmetric images.

In [3] and [32] a biological application of linear rules is presented. The authors refer to their algorithm as *Sweeper's Algorithm* and they claim that it is possible to simulate the process of chemotaxis by applying different basic rules at different sections of a lattice repeatedly at different times. As per the authors, this algorithm can also be used in various other applications like in informed search used in artificial intelligence, cryptography and pattern classification.



Figure 5.2: Edge detection using the Canny algorithm¹

5.3 Edge Detection with Cellular Automata

Digital image processing is an important task today. This field involves mathematical methods applied to an input image to produce a certain output image. This could be an image with enhanced visual quality (e.g. noise reduction) or some kind of feature detection.

Edge detection refers to mathematical methods for identifying sharp changes of brightness in an image. By using these methods, it is possible to identify shapes in a digital image for further processing. There are various algorithms for edge detection. One of the most prominent methods has been developed by JOHN CANNY and was first published in 1986 in [1]. Figure 5.2 shows an example of how the algorithm is able to detect edges in a digital image.

Some scientists proposed methods for several tasks in image processing by using cellular automata. An overview is provided in [20]. In [16] an approach is described, where linear rules are applied for edge detection. Some other applications of linear rules in image processing are presented in [31]. In [29] an approach using genetic algorithms was described.

¹JonMcLoone, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=44894482>



Figure 5.3: Edge detection using cellular automata[17]

5.3.1 Simple Approach using Linear Rules

In [16] and [18] the authors experimented with the linear rules described above to identify those suitable for edge detection in digital images. This method only works on binary images. Hence, it is necessary to first convert a digital image into a binary representation using an appropriate threshold value. The result is a binary matrix to which a given linear rule can be applied. In [17] the authors further elaborated this approach using a Moore-Neighbourhood with radius 2 and compared their results with existing methods for edge detection.

Figure 5.3 shows some results of the research outlined in [17]. The first picture shows the original image, the second is the binary image. The two images on the right-hand side show the edge detection results after the application of two different rules. The results are remarkable both in terms of the quality of the visual output and of the computation time.

5.3.2 Genetic Algorithms in Edge Detection using Cellular Automata

An interesting approach is used by THOMAS in [29], who developed a framework for identifying possible cellular automata rules suitable for edge detection and other image processing tasks using genetic algorithms.

Genetic algorithms are a form of evolutionary algorithms, that try to mimic the process of natural selection in nature to identify the method which is best suited to address

a certain task. These algorithms consist of a population of possible solutions to a problem each of which have certain properties, the chromosomes. In an iterative process the solutions are applied to a problem and the result is tested against a test function to evaluate the *fitness* of a solution. The best solutions are combined in a process known as *crossover* and are further affected by random *mutations* to generate a new generation of solutions. This is repeated to find an optimal solution.

This method is suitable for very large search spaces of possible solutions. We already saw that given a simple two dimensional cellular automaton with 2 states and a radius 1 Moore-Neighbourhood the number of possible rule functions exceeds the number of atoms in the universe. To find the best rule for a given problem, a genetic algorithm approach therefore seems well suited. The empirical results of this research can be found in [29].

5.4 Forest Fire Model

Another well known model which uses the concept of cellular automata is the so-called Forest Fire Model. It acts as a model for simulating the spread of forest fires in nature. There are many different types of this model varying in their complexity. Our discussion is based on [26].

The Forest Fire Model was initially proposed around 1990 independently by BAK ET AL., CHEN ET AL. as well as by DROSSEL ET AL. What makes this model interesting is that - although this has been subject to debates - it exhibits *self-organized criticality*. This means that the model produces a critical yet stable equilibrium state in which - in the long run - the average density of trees remains relatively stable.

This model uses a two dimensional grid as the underlying topology. Each cell on the

grid can take one of three states. We define our state set as follows:

$$\mathcal{S} = \{\text{tree}, \text{no-tree}, \text{tree-on-fire}\}$$

In the initial configuration the grid is randomly filled with `tree` and `no-tree` using a predefined density.

The state `tree` means that there is a tree on this field, `no-tree` that there is no tree and `tree-on-fire` means that the tree on this cell is on fire. Two parameters control the behaviour of this simple model. The parameter α is the probability that a `no-tree` cell becomes a `tree` cell in the next time step and the parameter γ is the probability for a `tree` cell to become a `tree-on-fire` cell in the next time step, which can be imagined as a lightning strike hitting that tree.

These parameters should obviously fulfil

$$0 < \gamma \ll \alpha < 1$$

which means that the probability for a lightning strike hitting a tree should be much smaller than the chance for a new tree to grow.

Let x_{ij} denote a cell on the grid. The transition function is defined as follows:

1. If x_{ij} is in state `no-tree` then the state of x_{ij} becomes `tree` with probability α . With probability $1 - \alpha$ the state remains the same.
2. If x_{ij} is in state `tree` and one or more cells in the Moore-Neighbourhood are in state `tree-on-fire`, then the state of x_{ij} becomes `tree-on-fire` in the next time step. If no cell in the neighbourhood is in state `tree-on-fire`, then with probability γ , x_{ij} changes its state to `tree-on-fire` due to a lightning strike. If no neighbours are on fire and no lightning strike occurs, the state remains unchanged.

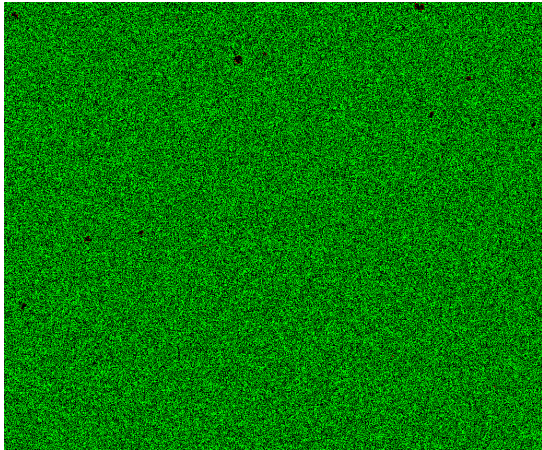


Figure 5.4: After 10 iterations

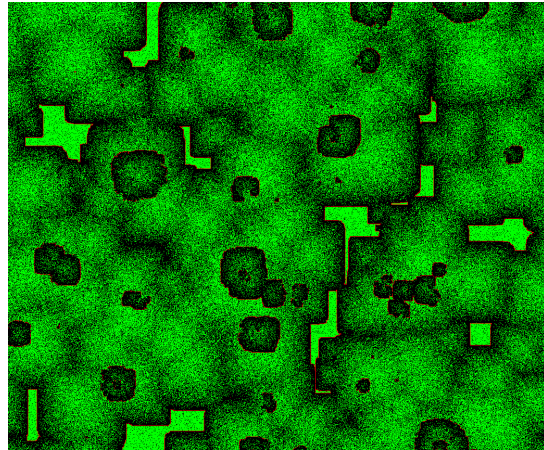


Figure 5.5: After 100 iterations

3. If the state of x_{ij} is `tree-on-fire`, then the fire extinguishes and the state becomes `no-tree` in the next time step.

Let us now look at an example implementation of the model. It uses about one million cells and an initial tree density of 0.4. We set $\alpha = 0.02$ and $\gamma = 0.000005$. In Figures 5.4, 5.5, 5.6 and 5.7 we see images of the model after different numbers of time steps. The state of `tree` is coloured in green, `no-tree` in black and `tree-on-fire` in red.

These images were calculated and illustrated in the browser using JAVASCRIPT and the canvas element in HTML5. The graphs in Figures 5.8 and 5.9 were dynamically calculated and visualized in the browser using Google Charts². There are many implementations of this model available online. To calculate our results, we slightly adapted the implementation that can be found [here](http://aryannayra.com/randomstuff/forestfire.html)³.

As we can see, the first 100 time steps make a big difference in the visual appearance of the model as the relative amounts of different states change fast during the first iterations. However, after a certain number of iterations, the visual appearance does not change any more. We can observe this by comparing Figures 5.6 and 5.7 to Figures

²<https://developers.google.com/chart>

³<http://aryannayra.com/randomstuff/forestfire.html>

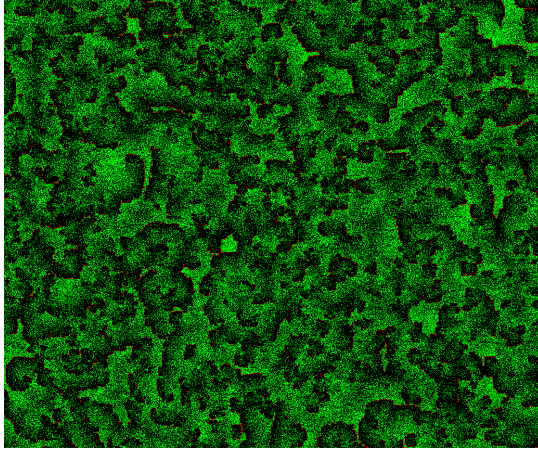


Figure 5.6: After 500 iterations

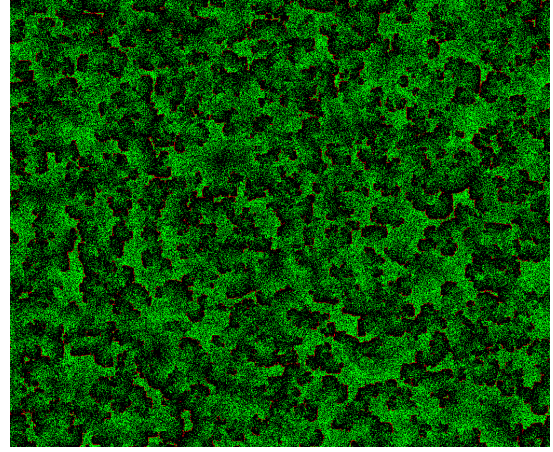


Figure 5.7: After 1000 iterations

5.4 and 5.5. This is due to the *self-organized criticality* of this system. However, the visual appearance does not say much about the system's actual properties. So let us calculate the density of trees and represent this quotient as a function of the time steps. In Figures 5.8 and 5.9 we can observe that the density of trees varies strongly in a first phase and later, after around 400 time steps stabilizes around 0.3. We can also observe that the stabilization effect correlates with the number of cells. The more cells used, the stronger the effect. Both calculations use an initial tree density of 0.4, but for the calculation in Figure 5.9 the number of cells was increased by the factor 10.

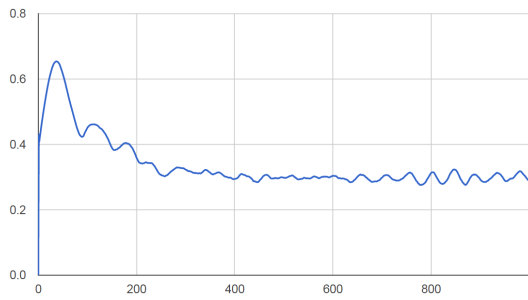


Figure 5.8: Forest Fire, 10^5 Cells

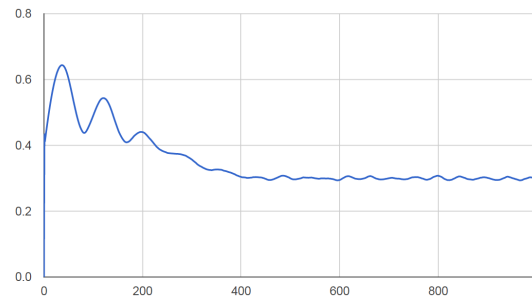


Figure 5.9: Forest Fire, 10^6 Cells

The stabilization that occurs in this models does not depend on the values of α and γ and can be observed for many different combinations.

State	Description
s_1	Street or area that cannot burn
s_2	Young tree or ashes
s_3	Growing tree
s_4	Old Tree
s_5	Fire almost extinguished
s_6	Fire burning
s_7	Fire just broke out

Table 5.2: State set

However, α and γ still have great influence on the behaviour of the model. For small α , we observe large fire sources spreading relatively fast. Since the trees grow back faster as α increases, smaller fire sources can be observed with larger values of α .

Due to its simplicity, this model is not actually suitable to be applied in real world simulations. However, it generates a basis on which more complex models can be developed. In [26] such an extended approach is presented.

First, we could introduce streets or other areas that cannot burn and therefore are capable of limiting the further spread of the fire. Additionally, in the model described above, a grown-up tree appeared out of nowhere with a certain probability. We could therefore introduce a way of simulating the growth phase of a tree and differentiate between young, growing and older trees. Although not used in the model described here, the different stages of a tree might have different burning properties (e.g. an older tree might have a higher chance to actually burn if a lightning strike occurs) as well. Finally, we might differentiate between several types of fires: a fire that just broke out, a fire that is almost extinguished and a fire that is burning normally. We see that this extended model uses ways to introduce several intermediate steps between the states of the first model.

This leads to the extended state set of Table 5.2.

Since we have a more complex state set in this model, we also need a more complex transition function. Let x_{ij} be a cell on the grid, then in this model the following update rule is used:

1. If x_{ij} is in state s_1 , it stays in this state all time.
2. If x_{ij} is in state s_2 , x_{ij} becomes s_3 with probability α . With probability $1 - \alpha$ nothing happens and the state remains s_2 .
3. If x_{ij} is in state s_3 , it becomes s_4 with probability α (it grows). If it does not grow (Probability $1 - \alpha$) and if any of its Moore-Neighbours is in state s_5 , s_6 or s_7 it becomes s_7 with probability γ due to a lightning strike. If it does not grow and no lightning strike occurs, it remains in state s_3 .
4. If x_{ij} is in state s_4 and at least one Moore-Neighbour is in state s_5 , s_6 or s_7 it becomes s_7 with probability β due to the spread of fire. If the fire does not spread the state becomes s_7 with probability γ due to a lightning strike.
5. If x_{ij} is in state s_5 it becomes s_2 in the next time step.
6. If x_{ij} is in state s_6 it becomes s_5 in the next time step.
7. If x_{ij} is in state s_7 it becomes s_6 in the next time step.

As we have more possible states, this model uses a more elaborate transition rule set. Most importantly, several factors limit the spread of fire in this model: Streets cannot burn and young trees need some time to grow before they can burn. So even though lightning strikes may occur on young trees, they do not burn before they grow older. Furthermore, another parameter β is introduced that determines the probability of fire spreading from a burning tree to a non-burning tree. This happened automatically in the simple model above, once any Moore-Neighbour of a cell was in burning state.

This assumption actually was quite unrealistic. An example simulation using this model is described in [26].

5.5 Lotka-Volterra Equations

In this section we present a cellular automaton that approximates solutions to the Lotka–Volterra equations. These are formulated as a system of first-order non-linear differential equations in two variables and are used to simulate predator-prey models in biology. These are models, in which the evolution of the size of two interacting populations is simulated.

However, these equations have also been used to simulate interactions between certain variables in economic models.

Consider a population of prey X and a population of predators Y . Interaction between X and Y takes place in the sense that X is the only food source for Y . The size of the predator population therefore depends on the available prey. Furthermore, the dynamics of the model are controlled through several parameters.

Definition 29. Lotka-Volterra Equations: For $t \in \mathbb{R}$ let $x(t)$ be the population of prey, $y(t)$ the population of predator and let $\alpha, \beta, \gamma, \delta \in \mathbb{R}^+$. The parameter α represents the reproduction rate of prey, β the mortality rate of prey, γ the mortality rate of predators if no prey is available and δ the reproduction rate of predators. The Lotka-Volterra equations describing the interaction of $x(t)$ and $y(t)$ for constant parameters $\alpha, \beta, \gamma, \delta \in \mathbb{R}^+$ have the following form:

$$\begin{aligned}\frac{\partial x}{\partial t} &= \alpha x - \beta xy \\ \frac{\partial y}{\partial t} &= \delta xy - \gamma y\end{aligned}$$

Usually it is supposed that the reproduction rates of populations only depend on their size, that no genetic adaptation occurs and that predators have limitless appetite.

However, many models have been introduced to simulate more complex and therefore realistic evolutions.

Solutions to the Lotka-Volterra equations are closed orbits [27] and it turns out that these solutions cycle periodically around the non-trivial fixed point of the equations, which is dependent on the parameters α, β, γ and δ . The fixed points can easily be calculated as follows:

$$0 = \alpha x - \beta xy$$

$$0 = \delta xy - \gamma y$$

$$\Rightarrow (x, y) = (0, 0) \text{ or } \left(\frac{\gamma}{\delta}, \frac{\alpha}{\beta}\right)$$

Figure 5.10 visualises a possible solution to these equations clearly showing the periodicity. It can also be seen that there is a phase shift between x and y .

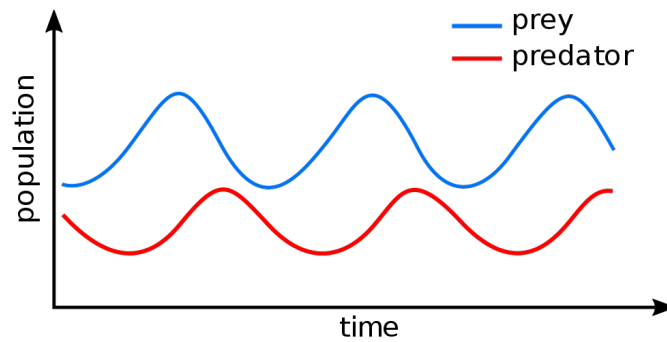


Figure 5.10: Solutions to the Lotka-Volterra equations⁴

⁴Author: AspidistraK, CC-BY-SA-4.0, <https://commons.wikimedia.org>

5.5.1 Approximating a Solution using a Cellular Automaton

In [26] a cellular automaton capable of modelling the dynamics of a predator-prey system as described by the Lotka-Volterra equations is proposed. It is also capable of approximating numerical solutions to these equations. This model uses a two dimensional grid and the Von-Neumann-Neighbourhood. In principle, each cell can take one out of three basic states: `predator`, `prey` or `void`. The latter state means that the cell is currently not occupied. However, some parameters need to be encoded in the states as well, resulting in a more complex state set. For this, a concept of *energy points* (EP) is used and three constants need to be defined in advance. This is shown in 5.3.

E_{prey}	Minimum EP needed for reproduction of a single prey cell
$E_{predator}$	Minimum EP needed for reproduction of a single predator cell
E_{gain}	EP for a predator after eating one prey cell

Table 5.3: Energy points

The cardinality of the state set is dependent on the values of E_{prey} and $E_{predator}$. For $E_{prey} = 3$ and $E_{predator} = 5$ we obtain four different `prey`-states and six different `predator`-states. Including the `void`-state we obtain a total of eleven different states for these values of E_{prey} and $E_{predator}$.

The state transition rule also uses a slightly adapted approach. In each time step, two transitions are applied. The first is called *Phase of Interaction* and the second is called *Phase of Development*.

In the **Phase of Interaction** the following rules are applied:

1. If x_{ij} is in void state, nothing happens.

2. If x_{ij} is in a `prey` state with energy points less than E_{prey} , x_{ij} moves to an arbitrary cell in its neighbourhood which is in `void` state. If no cell in this neighbourhood is in `void` state, nothing happens.
3. If x_{ij} is in a `prey` state with energy points equal to E_{prey} it reproduces itself to an arbitrary location in its neighbourhood. Both, the old and the new `prey`-cell obtain 0 energy points.
4. If x_{ij} is in a `predator` state and a minimum of one cell in its neighbourhood is in `prey`-state, x_{ij} 'eats' this cell, moves to its location leaving behind a `void`-cell and gains one energy point. If the cell has $E_{predator}$ energy points after the increase, it does not move, but generates another predator cell on the location of the prey cell. Both `predator` cells get a total of $\lfloor \frac{1}{2} E_{predator} \rfloor$ energy points.
5. If x_{ij} is in a `predator` state and no cell in its neighbourhood is in `prey` state, it moves to a cell which is in `void`-state in its neighbourhood. If all neighbouring cells are occupied by `predator` cells, nothing happens.

In the **Phase of Development** the following rules are applied:

1. If x_{ij} is in `void` state, nothing happens.
2. If x_{ij} is in a `prey` state with energy points less than E_{prey} , x_{ij} gains one energy point.
3. If x_{ij} is in a `prey` state with energy points equal to E_{prey} nothing happens.
4. If x_{ij} is in a `predator` state with a non-zero number of energy points, it loses one energy point.
5. If x_{ij} is in a `predator` state with no energy points, it changes its state to `void`.

Note that both phases could easily be described as one rule set since in the Phase of Development, the change of energy points for each cell does not depend on the states of the neighbouring cells. Therefore this model fits into our definition of a cellular automaton.

In [26] a sample implementation of this cellular automaton is presented and analysed and it turns out that - given appropriate values for the constants - this models generates results very similar to numerical solutions of the Lotka-Volterra equations obtained by the Runge-Kutta method.

CHAPTER 6 Conclusion

The goal of this thesis was to present an overview of theoretical aspects of cellular automata and to provide some examples of applications. In the first part, we introduced the concept by providing a general definition of cellular automata. We defined some important terms as well as several notions which were used throughout the whole thesis.

The second chapter introduced several important propositions in the realm of cellular automata. We identified the *global transition function* as the main object of interest and we introduced some concepts of the theory of dynamical systems. Furthermore, we elaborated a formal approach to analyse properties of the global transition function. This allowed us to present one of the most fundamental theorems for cellular automata, the Moore-Myhill Theorem. The relationship between injectivity and surjectivity allowed us to predict the existence of Garden-of-Eden configurations or equivalently the non-surjectivity and non-reversibility of the global transition function if we have some knowledge on some properties of this function.

The next chapter was about one dimensional cellular automata. We gave an overview of the work of STEPHEN WOLFRAM, who devoted a large part of his academic life to the analysis of Elementary cellular automata, for which we provided a definition and introduced the rule numbering scheme. We also presented some research on top of his theory which highlighted some shortcomings of his statistical classification. The most famous theoretical result in this context is the universality proof of rule 110 by Cook. After reviewing two examples of one dimensional cellular automata with extended neighbourhood, we proceeded to the two dimensional case.

We presented the well-known *Game of Life* and reviewed some theoretical aspects of this automaton. We saw an application of the Moore-Myhill Theorem, which allowed to predict the existence of orphan patterns because it is easily shown that the global transition function of the Game of Life is not injective. The second example was Langton's Vant, where we saw another construction to prove the universality of a system. Finally we presented the set of linear rules, since they offer some interesting possibilities for applications. This special class is also worth to mention because of its algebraic properties.

In the final chapter, we presented some applications of cellular automata. We reviewed some results in the realm of cryptography and we saw how Elementary cellular automata can be used for pseudo-random number generation. Then we discussed the idea of using reversible cellular automata to define an asymmetric encryption mechanism as done by KARI. We moved on to some applications in digital image processing, more precisely to edge detection and presented some research on how linear rules can be employed as edge detection algorithms for binary images. We also presented a model to simulate the spread of forest fires in nature and saw, how this model exhibits a form of self-organized criticality. Finally we presented a way to use cellular automata to simulate predator-prey models and approximate numerical solutions to non-linear differential equations, like the Lotka-Volterra equations.

Especially in the second half of the 20th century there was a large hype around cellular automata. The spread of personal computers which allowed to dynamically simulate the behaviour of these systems might have been a contributing factor to this interest. Also the extensive research conducted by WOLFRAM and his thoughts about a *New Kind of Science* based on cellular automata contributed to this popularity. In recent years, however, it seems that the concept forfeited some popularity in favour of other approaches, like agent-based modelling. The predator-prey model presented in this thesis, however, reveals how closely agent-based approaches are linked to the concept of cellular automata, or more precisely, how these approaches can be modelled as cellular automata.

As mentioned in [21], the concept of cellular automata might also be fruitfully applied in the search for alternatives to current hardware architectures due to their ability to fit into the scheme of massively parallel processing (MPP), an architecture where computations are distributed among a huge number of processors. Further progress in the field of quantum computing or the advent of new hardware architectures therefore might lead to another hype around cellular automata in the future. A versatile theory within this field, developed by countless contributors during the last decades, patiently awaits to be employed to solve problems, that we might not even be aware of – yet.

CHAPTER 7

Appendix

7.1 Time-Space Diagrams of Elementary Cellular Automata with Python

The following simple Python script was written to produce the time-space diagrams of one dimensional cellular automata in this thesis. The script is used as follows:

```
Usage: [filename].py [options]
```

Options:

```
-h, --help          show this help message and exit
-r RULENAME         rule number 0..255
-t TIMESTEPS        calculate <timesteps> timesteps
-s RANDOM_OR_NOT    firstline: 1 for random; 0 for single seed
-w WIDTH            matrix width
```

```
#!/usr/bin/env python
import random
import numpy as np
import matplotlib.pyplot as plt
import timeit
from optparse import OptionParser

T = 200

parser = OptionParser()
parser.add_option("-r", dest="RULENAME",
```

```
        help="rule_number_0..255")
parser.add_option("-t", dest="timesteps",
                  help="calculate_<timesteps>_timesteps")
parser.add_option("-s", dest="random_or_not",
                  help="firstline:_1_for_random;_0_for_single_seed")
parser.add_option("-w", dest="width", help="matrix_width")

(option, args) = parser.parse_args()

if not option.RULENAME:    # if RULENAME is not given
    RULENAME = 110
else:
    RULENAME = int(option.RULENAME)
if not option.timesteps:  # if timesteps is not given
    T = 200
else:
    T = int(option.timesteps)
if not option.random_or_not:  # if random_or_not is not given
    RANDOM_OR_NOT = 0
else:
    RANDOM_OR_NOT = int(option.random_or_not)
if not option.width:    # if width is not given
    n = 2 * T
else:
    n = int(option.width)

def initialize(randomfirstline):
    """initialize matrix"""
    matrix = np.zeros((T, n))
    if randomfirstline == 1:
        matrix[0] = np.random.randint(2, size=len(matrix[0]))
    else:
        matrix[0, int(n / 2)] = 1
```

```
    return matrix

def make_rule(rule_name, k):
    """returns rule name as list with 8 elements"""
    rule_name = int(RULENAME)
    rule_len = 2 ** k
    rule = list(map(int, bin(rule_name)[2:]))
    rule = [0] * (rule_len - len(rule)) + rule
    return rule

def populatematrix(matrix):
    """Populates the rest of matrix according to selected rule """
    for i in np.array(range(T - 1)) + 1:
        for j in range(n):
            row = matrix[i - 1]
            row = np.concatenate((row[-5:], row, row[:5]))
            o, p, q, r, s = row[5 + j - 2:5 + (j + 2) + 1]
            if p == 1 and q == 1 and r == 1:
                matrix[i, j] = make_rule(RULENAME, 3)[0]
            elif p == 1 and q == 1 and r == 0:
                matrix[i, j] = make_rule(RULENAME, 3)[1]
            elif p == 1 and q == 0 and r == 1:
                matrix[i, j] = make_rule(RULENAME, 3)[2]
            elif p == 1 and q == 0 and r == 0:
                matrix[i, j] = make_rule(RULENAME, 3)[3]
            elif p == 0 and q == 1 and r == 1:
                matrix[i, j] = make_rule(RULENAME, 3)[4]
            elif p == 0 and q == 1 and r == 0:
                matrix[i, j] = make_rule(RULENAME, 3)[5]
            elif p == 0 and q == 0 and r == 1:
                matrix[i, j] = make_rule(RULENAME, 3)[6]
            elif p == 0 and q == 0 and r == 0:
                matrix[i, j] = make_rule(RULENAME, 3)[7]
```

```
    return matrix

def main():
    """Main Method"""
    start = timeit.default_timer()
    matrix = initialize(RANDOM_OR_NOT)
    resultmatrix = populatematrix(matrix)
    plt.rcParams['figure.figsize'] = [10, 20]
    plt.subplots_adjust(0.0, 0.03, 1, 0.98, 0.0, 0.0)
    figManager = plt.get_current_fig_manager()
    figManager.window.showMaximized()
    plt.axis('off')
    plt.imshow(resultmatrix, interpolation='nearest',
               filternorm=1, cmap=plt.cm.gray)
    stop = timeit.default_timer()
    print('Calculation_done_in:', stop - start, 'seconds')
    plt.show()

if __name__ == '__main__':
    main()
```

List of Figures

1.1	Von-Neumann	5
1.2	Moore	5
1.3	Periodic boundary conditions	6
2.1	Possible orbits	10
3.1	Wolfram classification scheme	22
3.2	Rule 90	23
3.3	Rule 73	23
3.4	Rule 30	23
3.5	Rule 110	23
3.6	Rule 182	23
3.7	Rule 101	23
3.8	Rule 90, 5 time steps	24
3.9	Rule 90, 50 time steps	24
3.10	Rule 90, 100 time steps	25
3.11	Geometric construction of rule 90	25
3.12	Rule 150, 1000 time steps	26
3.13	Geometric construction of rule 150	26
3.14	Rule 90, 5 time steps	27
3.15	Rule 90, 50 time steps	27
3.16	Rule 90, 100 time steps	27
3.17	Conus textile	27
3.18	Class 1: Rule 160	29
3.19	Class 2: Rule 108	29
3.20	Class 3: Rule 126	29

3.21	Class 4: Rule 110	29
3.22	Rule 28	33
3.23	Rule 220	34
3.24	Rule 110, 10 time steps	38
3.25	Rule 110, 100 time steps	38
3.26	Rule 110, 500 time steps	39
3.27	Rule 110, 500 time steps, random initial condition	41
3.28	300 time steps	43
3.29	300 time steps	44
3.30	Two stable structures	45
3.31	Other stable structures	45
3.32	Sad meeting	45
3.33	Sad meeting 2	45
3.34	300×300	47
3.35	300×300 , random	47
3.36	300×301	47
3.37	30×23	48
3.38	60×47	48
3.39	$n = 767$	50
3.40	$n = 7$	51
3.41	$n = 17$	51
3.42	$n = 8$	51
3.43	$n = 4$	51
4.1	Five steps in the Game of Life	56
4.2	Some common invariant forms	56
4.3	Some period 2 oscillators	57
4.4	The CP-Pulsar	57
4.5	The glider	58
4.6	A glider gun	58
4.7	First orphan	60
4.8	Current record	60

4.9	After 500 iterations	63
4.10	After 3000 iterations	63
4.11	After 10000 iterations	63
4.12	After 10360 iterations	63
4.13	After 13842 iterations	64
4.14	Logical gates in Vant	66
4.15	Highway constructor	66
5.1	Rule 30, 300 time steps	73
5.2	Canny edge detection	77
5.3	Edge detection using cellular automata	78
5.4	After 10 iterations	81
5.5	After 100 iterations	81
5.6	After 500 iterations	82
5.7	After 1000 iterations	82
5.8	Forest Fire, 10^5 Cells	82
5.9	Forest Fire, 10^6 Cells	82
5.10	Solutions to the Lotka-Volterra equations	86

List of Tables

3.1	Period length	51
3.2	Period length 2	51
5.1	Image translation	76
5.2	State set	83
5.3	Energy points	87

Bibliography

- [1] **Canny, J.:** *A Computational Approach to Edge Detection*, In: IEEE Transactions on Pattern Analysis and Machine Intelligence archive, Volume 8, Issue 6, pp. 679-698, IEEE Computer Society Washington, DC, USA, 1986
- [2] **Ceccherini-Silberstein, T. et al.:** *Cellular Automata and Groups*, Springer Berlin Heidelberg, 2010
- [3] **Choudhury, P. et al.:** *Theory and Applications of Two-dimensional, Null-boundary, Nine-Neighborhood, Cellular Automata Linear rules*, arXiv:0804.2346, 2008
- [4] **Cook, M.:** *Universality in Elementary Cellular Automata*, In: Complex Systems 15, pp. 1-40 , California, 2004
- [5] **Culik K. et al.:** *Undecidability of CA classification schemes*, In: Complex Systems 2, pp. 177-190, 1988
- [6] **Gajardo, A. et al.:** *Complexity of Langton's ant*, In: Discrete Applied Mathematics 117, pp. 41-50, 2002
- [7] **Kari, J.:** *Reversibility of 2D cellular automata is undecidable*, In: Physica D: Non-linear Phenomena, Vol. 45, Issues 1–3, pp. 379-385, 1990
- [8] **Kari, J.:** *Theory of cellular automata: A survey*, Department of Mathematics, University of Turku, 2004

- [9] **Kari, J.:** *Reversibility and surjectivity problems of cellular automata*, In: Journal of Computer and System Sciences 48, pp. 149-182, 1994
- [10] **Kari, J.:** *Reversible cellular automata*, In: Lecture Notes in Computer Science 3572, pp. 2-23, Springer, 2005
- [11] **Kari, J.:** *Cellular Automata*, University of Turku, 2016
- [12] **Koc, C.K. et al.:** *Inversion of cellular automata iterations*, In: IEE Proceedings of Computers and Digital Techniques 144, pp. 279-284, 1997
- [13] **Langton, C.:** *Studying artificial life with cellular automata*, In: Physica 22D, pp. 120-149, 1986
- [14] **Macfarlane, A.J.:** *Generating functions for integer sequences defined by the evolution of cellular automata with even rule numbers*, Centre for Mathematical Sciences, D.A.M.T.P., Cambridge, 2016
- [15] **Meier, W. et al.:** *Analysis of Pseudo Random Sequences Generated by Cellular Automata*, In: Davies D.W.: Advances in Cryptology - EUROCRYPT 1991. Lecture Notes in Computer Science, Vol. 547, Springer, Berlin, Heidelberg, 1991
- [16] **Mohammed, J. et al.:** *An Efficient Edge Detection Technique by Two Dimensional Rectangular Cellular Automata*, arXiv:1312.6370, 2013
- [17] **Mohammed, J. et al.:** *A Cellular Automata based Optimal Edge Detection Technique using Twenty-Five Neighborhood Model*, In: International Journal of Computer Applications, Vol. 84, pp. 27-33, 2013
- [18] **Nayak, D. et al.:** *A Survey on Two Dimensional Cellular Automata and Its Application in Image Processing*, arXiv:1407.7626, pp. 177-190, 2014

- [19] **Neumann, J.:** *Theory of self-reproducing automata*, University of Illinois Press, Urbana and London, 1966
- [20] **Popovici, D. et al.:** *Cellular automata in image processing*, Fifteenth International Symposium on Mathematical Theory of Networks and Systems, Vol. 1, 2002
- [21] **Renner, U.:** *Zelluläre Automaten – Ein ausgewählter Überblick*, In: W. Eisenberg et al.: Synergie, Syntropie, nichtlineare Systeme, Heft 3, Soft computing – Kuriosa, Verlag im Wissenschaftszentrum Leipzig, 2000
- [22] **Röck, A.:** *Pseudorandom Number Generators for Cryptographic Applications*, Diplomarbeit, Paris-Lodron-Universität Salzburg, Salzburg, 2005
- [23] **Salman, K.:** *Analysis of Elementary Cellular Automata Boundary Conditions*, In: International Journal of Computer Science and Information Technology (IJCSIT), Vol. 5, No. 4, Murfreesboro, 2013
- [24] **Santos, T.:** *Cellular Automata and Cryptography*, Master Thesis, Faculdade de Ciências da Universidade do Porto em Ciência de Computadores, 2014
- [25] **Schiff, J.L.:** *Introduction to Cellular Automata*, online: http://psoup.math.wisc.edu/pub/Schiff_CAbook.pdf
- [26] **Scholz, D.:** *Pixelspiele - Modellieren und Simulieren mit zellulären Automaten*, Springer Spektrum, Springer-Verlag Berlin Heidelberg 2014
- [27] **Steiner, A. et al.:** *Parametrische Lösungen der Räuber-Beute-Gleichungen im Vergleich*, In: Il Volterriano, No. 7, pp. 32-44, 1999
- [28] **Sutner, K.:** *A Note on Culik-Yu Classes*, In: Complex Systems 3, pp. 107-115, 1989

- [29] **Thomas, C.:** *Evolution of Cellular Automata for Image Processing*, Thesis, School of Computer Science, University of Birmingham (UK), 2000
- [30] **Tomassini, M. et al.:** *Cryptography with cellular automata*, In: Applied Soft Computing 1, pp. 151-160, 2001
- [31] **Uguz, S. et al.:** *2D Cellular Automata with an Image Processing Application*, In: Proceedings of the 3rd International Congress APMAS2013, Acta Physica Polonica A, Vol. 125, No. 2, 2014
- [32] **Utkaleswar, P. et al.:** *Sweeper's Algorithm and its Application on Image Clustering*, In: International Journal of Computer Applications, Vol. 119, No. 8, 2015
- [33] **Weisstein, E.:** *Jacobsthal Number*, A Wolfram Web Resource, <http://mathworld.wolfram.com/JacobsthalNumber.html>
- [34] **Weisstein, E.:** *Rule 30*, A Wolfram Web Resource, <http://mathworld.wolfram.com/Rule30.html>
- [35] **Weisstein, E.:** *von Neumann Neighborhood*, A Wolfram Web Resource, <http://mathworld.wolfram.com/vonNeumannNeighborhood.html>
- [36] **Weisstein, E.:** *Moore Neighborhood*, A Wolfram Web Resource, <http://mathworld.wolfram.com/MooreNeighborhood.html>
- [37] **Wilensky, U.:** *NetLogo Vants model*. <http://ccl.northwestern.edu/netlogo/models/Vants>, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL
- [38] **Wilensky, U.:** *NetLogo*. <http://ccl.northwestern.edu/netlogo/>,

Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL

- [39] **Wolfram, S.:** *A New Kind of Science*, Wolfram Media, Champaign, IL, 2002
- [40] **Wolfram, S.:** *Cellular Automata and Complexity. Collected Papers*, Perseus Books Group, 1994
- [41] **Wolfram, S.:** *Cellular automata*, In: *Cellular Automata and Complexity. Collected Papers*, Perseus Books Group, 1994
- [42] **Wolfram S.:** *Cryptography with Cellular Automata*, In: *Cellular Automata and Complexity. Collected Papers*, Perseus Books Group, 1994
- [43] **Wolfram, S.:** *Statistical mechanics of cellular automata*, In: *Rev. Mod. Phys.* 55, 1983
- [44] **Wolfram, S.:** *Theory and Applications of Cellular Automata*,, World Scientific Press, Singapore, 1986
- [45] **Wolfram, S.:** *Universality and complexity in cellular automata*,, *Physica 10 D*, 1984

Abstract

The concept of a cellular automaton is quite simple. The basis is a d -dimensional grid, consisting of a finite or infinite number of cells. Each cell can take one state out of a finite state set in each time step. Every element of the cell space is mapped onto an element of the state set. Mathematically, this is a function and we call this function the *configuration* of the automaton at a given time. In discrete time steps, each cell changes its state depending on the states of its neighbours in a predefined neighbourhood, according to a certain update function. The arguments of this update function are the states of the neighbouring cells, which might include the state of the cell itself as well. Through this mapping, we obtain a new configuration. The function that globally maps one *configuration* to another one is called the *global transition function* and constitutes the main object of interest when theoretically analysing these systems. By investigating the properties of this function, we can answer certain questions about the behaviour of the automaton. They can be analysed with regards to basic mathematical properties like injectivity and surjectivity. By doing so, we may ask if the automaton is reversible, meaning that the global transition function has an inverse function. We can also ask questions like: Will the system ever reach a certain configuration? Similarly, we may ask such questions about the behaviour of certain subsets of the cell space as well. A variety of methods might be used to answer such questions, including the theory of discrete dynamical systems or methods from theoretical computer science, like algorithmic decidability and complexity theory.

Cellular automata were introduced by John von Neumann and Stanislaw Ulam around 1940. Neumann wanted to show that self-replicating machines are theoretically possible and worked on such models. Following an advice of Ulam, Neumann developed the idea of cellular automata. In the following decades this concept became widely

known and inspired the works of many scientists. During this time, many models with interesting properties were developed. All of these models share one basic principle: the simplicity of their construction. Yet, the dynamics of these models yield highly complex behaviour in their time-space evolution. Scientists from many different disciplines were therefore interested in understanding the behaviour of these systems. Not only in mathematics and computer science, but also in other disciplines like physics, biology and even philosophy many researchers were fascinated by this topic. But this interest was not limited to the scientific community. Some cellular automata like the Game of Life and others nowadays have a worldwide fan base of individuals, who share the fascination for these *miracles of simplicity*.

Cellular automata are interesting in many ways. The definitions of these models are often blueprints for their implementation. The work on this thesis started by implementing a great variety of models and analysing their behaviour. This allows to investigate some changes in the behaviour when certain parameters are adjusted. In mathematics, however, we often ask questions that cannot be answered by simply watching the behaviour of a model on a computer screen. Therefore, to really understand the behaviour of a model, one needs to understand the underlying theory and see, which methods might help to answer certain questions. In this manner, a theoretical result, the Moore-Myhill Theorem, allows to predict configurations on subsets of the cellular space in the Game of Life, that actually can never be reached. By knowing this, the search for such configurations becomes meaningful and kept researcher busy during the last decades. On the other hand this research in turn revealed theoretical results again, like the identification of a lower bound for the dimension of such configurations.

This thesis summarizes the theoretical background of this topic and specifically aims to present a compact overview of theoretical properties of one and two dimensional cellular automata. There is, however, nothing more practical than a good theory and therefore the theoretical work on the subject during the last decades revealed many possible applications in turn. The research of J. Kari in [7] are a good example how

this interplay works. Even though the focus of this thesis clearly lies on the theory, we also provide many examples of possible applications of cellular automata in a variety of fields. We therefore present such ideas in the realm of cryptography, digital image processing and within the simulation of complex systems.

Kurzfassung

Das Modell des zellulären Automaten ist sehr einfach: Basis ist eine d -dimensionale Anordnung von unendlich oder endlich vielen Zellen, die jeweils gewisse Zustände annehmen können. Die möglichen Zustände können als endliche Menge aufgefasst werden. Jedem Element des Zellularraumes wird ein Zustand, also ein Element aus der Zustandsmenge, zugewiesen, weshalb es sich hierbei um eine Funktion von der Menge der Zustände auf die Menge der Zellen handelt. Wir sprechen hierbei von der *Konfiguration* des Automaten zu einem gegebenen Zeitpunkt. In diskreten Zeitschritten verändern die Zellen ihre Zustände abhängig von den Zuständen der Zellen in einer vorab festgelegten Nachbarschaft basierend auf einfachen Regeln. Zu jedem Zeitpunkt erhält man so eine neue Konfiguration des Automaten. Die Funktion, die eine Konfiguration zum Zeitpunkt t in eine neue Konfiguration zum Zeitpunkt $t + 1$ überführt, heißt die globale Überföhrungsfunktion (*global transition function*) und bildet das zentrale Objekt in der Erforschung derartiger Systeme. Diese Funktionen können zum Beispiel hinsichtlich elementarer Eigenschaften von Abbildungen wie etwa Injektivität und Surjektivität untersucht werden. In weiterer Folge kann gefragt werden, ob die Funktion eine Inverse besitzt, der Automat also reversibel ist. Oder wir können uns fragen, ob diese oder jene Konfiguration jemals angenommen wird, oder welche genauen Bedingungen für ein bestimmtes Verhalten verantwortlich sind. Diese Fragen können natürlich auch bezüglich gewisser Teilmengen des Zellularraumes gestellt werden. Um derartige Fragen zu beantworten, können verschiedene mathematische Methoden zum Einsatz kommen. Neben allgemeinen Grundlagen spielt die Theorie diskreter dynamischer Systeme oder auch Methoden der theoretischen Informatik eine wichtige Rolle, etwa wenn es um Fragen der algorithmischen Entscheidbarkeit geht.

Die Erfindung des Konzepts des Zellularautomaten wird gemeinhin John von Neumann zugeschrieben, dessen Ziel es war, selbstreplizierende Maschinen zu konzipieren. Seine Motivation war es, zu zeigen, dass derartige Maschinen theoretisch möglich sind. Stanislaw Ulam lieferte ihm hierzu wesentliche Ideen, die in der Folge zu seiner Entwicklung des Konzepts des Zellularautomaten beitrugen. In den folgenden Jahrzehnten erlangte diese Idee große Berühmtheit und viele Arbeiten beschäftigten sich mit Fragen in diesem Zusammenhang. Zudem wurden immer mehr Automaten konzipiert, die äußerst interessante Eigenschaften aufwiesen. Ein gemeinsames Merkmal all dieser Systeme ist die Einfachheit ihrer Konstruktion und die Komplexität ihrer Dynamik. Trotz weniger und sehr einfacher Regeln sowie einer streng deterministischen Entwicklung führen derartige Systeme zu beeindruckender Komplexität. Wissenschaftlerinnen und Wissenschaftler aus vielen Disziplinen haben sich daher mit diesem Thema beschäftigt. Nicht nur im Bereich der Mathematik und der Informatik, sondern auch im Bereich der Physik, der Biologie bis hin zur Philosophie begeistern diese Systeme bis heute viele Forscherinnen und Forscher. Aber dieses Interesse beschränkt sich nicht nur auf die Wissenschaft. Rund um einige zelluläre Automaten haben sich weltweit regelrechte Fangemeinden gebildet, die die Faszination dieser *einfachen Wunder* teilen.

Es ist stets erstaunlich, wenn einfache Prozesse komplexe Entwicklungen zeigen. Dieses Phänomen kann häufig im Bereich diskreter dynamischer Systeme beobachtet werden. Viele Menschen sind von der Schönheit von fraktalen Strukturen erstaunt, oder von dem Moment, ab dem bei der Veränderung gewisser Parameter die logistischen Gleichung die Grenze zwischen Ordnung und Chaos berührt. Oft rührt diese Faszination von visuellen Darstellungen des Verhaltens derartiger Modelle. Die wirkliche Schönheit dieser einfachen Wunder offenbart sich häufig jedoch erst durch die zugrunde liegende Theorie und die Details ihrer Konstruktion.

Zelluläre Automaten sind in vielerlei Hinsicht interessant. Die Definitionen einiger Automaten laden dazu ein, selbst an Implementierungen zu arbeiten. Auch die Beschäftigung mit dem Thema im Rahmen dieser Arbeit begann damit, dass eine Vielzahl

an Automaten implementiert wurde und durch Veränderung gewisser Parameter ihr Verhalten untersucht wurde. In der Mathematik stellen wir jedoch häufig Fragen, deren Beantwortung nicht dadurch möglich ist, das Verhalten eines Modells am Bildschirm zu beobachten. So ist für ein tiefer gehendes Verständnis dieser Modelle eine Beschäftigung mit der zugrunde liegenden Theorie unabdingbar. Oft zeigen theoretische Resultate auch erst, wo es sich lohnt, durch praxisbezogene Forschung näher hinzusehen. So erlaubt etwa ein theoretisches Resultat, das Moore-Myhill-Theorem, die theoretische Vorhersage der Existenz von Konfigurationen auf Teilmengen des Zellularraumes im Game of Life, die niemals entstehen können. Dadurch macht die praktische Suche nach derartigen Konfigurationen erst Sinn. Durch diese Forschung konnten wiederum theoretische Resultate geliefert werden, wie etwa eine untere Schranke für die Dimension solcher Konfigurationen.

Diese Arbeit fasst den theoretischen Hintergrund dieses Themas zusammen und hat zum Ziel, die wichtigsten Resultate in diesem Bereich in kompakter Form darzustellen. Aber es gibt - wie man sagt - nichts Praktischeres als eine gute Theorie und daher hat auch die theoretische Forschung auf diesem Gebiet in den letzten Dekaden eine Vielzahl möglicher Anwendungsfelder identifiziert. Die Arbeiten von J. Kari in [7] sind ein gutes Beispiel für dieses Wechselspiel von Theorie und Praxis. Auch wenn der Fokus dieser Arbeit klar auf theoretischen Aspekten dieser Modelle liegt, so sollen auch viele Beispiele für Anwendungen in verschiedenen Gebieten beschrieben werden. Wir beschränken uns hierbei auf Anwendungsmöglichkeiten im Bereich der Kryptographie, der digitalen Bildverarbeitung und der Simulation komplexer Systeme.