Modelling with cellular automata: problem solving environments and multidimensional applications
Naumov, L.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 3. Exploring Evolving 1D Structures and 2D Universal Data Indexing

This chapter presents two applications of CAME&L in theoretical research. First we will perform the classification of all structures generated by 1D binary cellular automata from a single seed. Then we will present methods of a universal state data indexing for different cellular automata grids. We propose a set of methods of such generalized coordinates for all the possible two-dimensional grids, composed of regular polygons.

3.1 Classification of Structures Generated by 1D Binary Cellular Automata from a Single Seed

All the studies here are performed with the help of a one-dimensional binary cellular automaton. If this automaton is denoted as $A = \{G, Z, N, f\}$, then

- $G$ is a one-dimensional space of cells with Cartesian coordinates and periodic boundary conditions;
- $Z = \{0, 1\}$;
- $N(c_i) = \{c_{i-1}, c_i, c_{i+1}\}$, where $c_k$ is an $k$-th cell on the grid;
- Using upper indices to designate the timestep and lower indices to define a cell’s coordinate the update rule reads $f(c_{i-1}^t, c_i^t, c_{i+1}^t) = c_{i}^{t+1}$.

All 256 possible structures generated by such one-dimensional binary cellular automata containing a single seed at step zero are discussed at large in [84, 85]. By a term “structure”, we mean a sequence of states of a one-dimensional grid in time, assembled into a two-dimensional lattice one under the other. Some of these structures are equivalent with respect to invariant operations such as mirror reflection, inversion, offsets and so on.

---

This section is partly based on: L. Naumov and A. Shalyto. Classification of Structures Generated by One-Dimensional Binary Cellular Automata from a Point Embryo. Journal of Computer and Systems Sciences International. Vol. 44/5. pp 137–145. 2005. Corresponding research was supported by the Russian Foundation for Basic Research, project no. 05-07-90086.
The purpose of the research in this section is to classify structures generated by a one-dimensional binary cellular automata\(^\ast\) from a single seed with respect to the invariant operations introduced and to calculate the number of representatives in various classifications.

An approach to cellular automata behaviour classification was offered in [84]. In [40,86,87] a method of distinguishing cellular automata with “interesting” behaviour was offered. Here we will concentrated on calculating the exact number of different observed structures, to illustrate the complexity of possible behaviour of such trivial systems with the exact numbers. Most “interesting” structures are unique in the class and can be produced by single transition function. Approaching different classifications we will try to minimize the number of distinguished classes. Nevertheless this number should reflect the amount of possible structure shapes in a detailed manner (not four classes as in [84] and not just two as in [87]). In addition we want to show that there are automata with cells with memory and with memoryless cells, which produce structures of the same class.

Elementary one-dimensional automata are able to generate self-reproducing configurations such as the Sierpinsky gasket and the binary Pascal triangle. In [88,89] these structures are obtained by rather elaborate mathematical methods. One-dimensional cellular automata are simpler tools for generating structures like these. We will consider classes of such structures separately.

### 3.1.1 Specification of the Transition Function

As was mentioned above, we consider cellular automata whose neighbourhood pattern consists of the right and left neighbours of a cell. Here we will study the automaton behavior for a number of steps significantly smaller than the grid width. Therefore the boundary conditions do not show up.

The transition function of the automaton, which determines the evolution of the state of a cell can be best specified in tabular form. The left three columns of such table contain the current states of cells \(i-1\), \(i\), and \(i+1\) arranged in lexicographical order. The rightmost column stores the next state of cell \(i\).

This table has eight rows, each containing one of the two possible values of the next state. This means that there are 256 various transition functions and, hence, 256 cellular automata for arbitrary initial conditions. As was proposed in [84], each automaton can be associated with a number in the range from 0 to 255 whose binary representation appears in the rightmost column of the transition table. This enumeration is well known, and first introduced by Wolfram.

In the following discussion we will use the language of Boolean expressions, where \(\oplus\) designates exclusive disjunction, \(\lor\) – inclusive disjunction, and overlining means logical inversion.

For example, function number \(90 = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 1 \cdot 2^4 + 0 \cdot 2^5 + 1 \cdot 2^6 + 0 \cdot 2^7\) has the form

\[
c_i^{t+1} = c_{i-1}^{t} \oplus c_{i+1}^{t}.
\] (3.1)

\(^\ast\)In this section saying “cellular automaton” we will mean exactly such special kind of automaton.
Note that this function determines a memoryless automaton, because the right-hand side of the corresponding formula is free of variable $c_t^i$.

Function number $165 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 + 0 \cdot 2^6 + 1 \cdot 2^7$ can be implemented by formula

$$c_{t+1}^i = \overline{c_{t-1}^i} \lor c_{t+1}^i. \quad (3.2)$$

This function is the inverse of the function number 90. This fact is evidenced by the equality $90 + 165 = 255$.

### 3.1.2 Initial Conditions

The structure generated by a cellular automaton with a fixed transition function is defined by its initial state. Unlike Wolfram, we study only the situation where each automaton at step zero contains a single cell in state “1”, situated in the center of the grid. All other cells contain “0”.

### 3.1.3 Comparison of Grids’ States as a Basis of Classifications

A basic operation for a classification of structures, produced by cellular automata, is a comparison of their states. We will consider this comparison according to operations of invariance. In general terms this means that if structure $a$ and structure $b$ belong to a single class then

$$\forall t \ I(S_a(t), S_b(t)), \quad (3.3)$$

where $t$ is a timestep, $S_a(t)$ and $S_b(t)$ are states for both structures on the $t$-th timestep, and $I$ is a predicate, which is true if both argument states are invariant with respect to a current operation of invariance. To make this notation totally clear we should write also that the cell’s state

$$S(t) = \{c_{t-N}, c_{t-N+1}, \ldots, c_{t-1}, c_t^0, c_t^1, \ldots, c_{t-N-1}, c_{t+N}\}, \quad (3.4)$$

where, consequently, the total amount of cells on the grid is $2N + 1$.

The theoretical statement of invariance, which is posed by Eq. 3.3, is hardly usable in practice. It is not possible to check the invariance predicate for any $t$, but only for a finite set. Here we will assume that, if two cellular automata exhibit the required behavior in first $\tilde{t}$ steps, then they continue exhibiting it in the sequel. A proof of the correctness of this assumption follows.

**Proof:**

Let us assume that we are testing a hypothesis that structures, produced by automata $a$ and $b$, belong to one class. They have the same initial conditions (in other words, sets $S_a(0)$ and $S_b(0)$ are equal), consequently, it is possible to perform the proof by induction.

Let us assume that
∀ \( t \leq \tilde{t} \) \( I(S_a(t), S_b(t)). \) \hfill (3.5)

We need to prove that

\[
I(S_a(\tilde{t} + 1), S_b(\tilde{t} + 1)). \hfill (3.6)
\]

The predicate stated by Eq. 3.6 is true if all eight possible combinations of cell’s neighbourhoods (or all eight possible transitions ways) were already met during an automaton progress within \( \tilde{t} \) timesteps.

A set of three values of neighbouring cell states from the previous timestep, which lead to a distinct state of the cell on a current timestep, will be called a prototype of a cell. In Figure 3.1 the structure generated by automaton number 165 is presented in textual form after eight steps.

<table>
<thead>
<tr>
<th>Timestep</th>
<th>Textual Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1</td>
</tr>
<tr>
<td>2</td>
<td>1 1 1 1 1 0 1 1 1 0 1 1 1 1 1</td>
</tr>
<tr>
<td>3</td>
<td>1 1 1 1 0 0 1 0 0 1 0 0 0 1 1 1</td>
</tr>
<tr>
<td>4</td>
<td>1 1 1 0 1 1 1 1 1 1 0 1 1 1 1</td>
</tr>
<tr>
<td>5</td>
<td>1 1 0 1 0 1 1 1 1 1 0 1 0 1 1</td>
</tr>
<tr>
<td>6</td>
<td>1 0 1 1 1 0 1 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>7</td>
<td>0 1 0 1 0 1 0 1 0 1 0 1 0 1 0</td>
</tr>
</tbody>
</table>

Figure 3.1: Structure, generated by function 165 in textual form after 8 steps. The subset of cells, which will be considered, are marked with black.

If we will consider the center cells from the third timestep (marked with black in Fig. 3.1) then prototypes of all cells will be found on the second timestep as overlapping triplets. In Figure 3.2 prototypes are shown as black triplets.

<table>
<thead>
<tr>
<th>Timestep 3</th>
<th>Textual Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 1 0 1 0 1 0 1 1</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>2</td>
<td>1 0 1 1 0 1 1 0</td>
</tr>
<tr>
<td>3</td>
<td>0 1 0 1 0 1 0 1</td>
</tr>
</tbody>
</table>

Figure 3.2: Prototypes for the center cells from the third timestep of the structure, generated by function 165.

For given initial conditions we have to show that there should be a timestep \( \tilde{t} \), after which all possible prototypes would be met to prove that the predicate stated by Eq. 3.6 is true. In almost half of cases (118 automata out of 256) all eight prototypes appear while an automaton is progressing, but for the rest we have to receive evidence that all prototypes, which may appear in principle, would be met within \( \tilde{t} \) timesteps. For example, the progress of automaton number 0 will involve only four prototypes: “000”, “001”, “010”, and “100”. All of them appear on the very first timestep.
3.1. Classification of Structures Generated by 1D Binary Cellular Automata from a Single Seed

An exhaustive computational study shows that all 256 automata, considered here, involve all possible prototypes before the fifth timestep. The following four sets of automata should be considered separately.


- The rest of the automata are generating less than eight prototypes, but the whole subset of occured prototypes are observed before the fifth timestep.

- \{1, 2, 3, 6, 10, 11, 14, 16, 17, 20, 23, 24, 28, 31, 33, 34, 35, 38, 42, 43, 46, 48, 49, 50, 52, 56, 58, 63, 55, 66, 70, 74, 80, 81, 84, 87, 88, 95, 98, 106, 112, 113, 114, 116, 119, 120, 122, 127, 130, 134, 138, 139, 148, 152, 156, 162, 166, 170, 171, 174, 176, 178, 180, 184, 186, 194, 198, 202, 206, 208, 209, 212, 216, 220, 222, 226, 234, 238, 240, 241, 242, 244, 248, 250, 252, 254\} - demonstrate static behaviour, such as oscillating (like automaton number 1), moving (like automaton number 2 or 9), or expanding, but saving the pattern (like automaton number 58).

- \{18, 26, 82, 90, 146, 154, 210, 218, 129, 161\} - demonstrate self-reproductive behaviour.

Showing that there is a timestep \(\bar{t}\), before which all possible prototypes were met for any given automaton, we have proven that the predicate, stated by Eq. 3.6, is true.

3.1.4 Invariance with Respect to the Operation “Equality”

The same structure, produced by cellular automata with different transition functions, represents a basic criterion for classifying them together. In this case we may speak about the invariance with respect to the “equality” operation. The same class may contain cellular automata having essentially different transition functions.

For instance, trivial behavior (the seed dies at the very first step) is exhibited by automata with different functions such as “identical zero” and “two or more out of three”. The whole class consists of 16 functions: 0, 8, 32, 40, 64, 72, 96, 104, 128, 136, 160, 168, 192, 200, 224, 232. It is characteristic of these functions that the rightmost columns of their transition tables have zero in the rows whose left part
contains at most a single unit. The transposed columns of their values have the form 
\[0 0 ? 0 ? ? ?]^T\], where the symbol “?” can be either “0” or “1”.

There are 143 invariance classes with respect to the “equality” operation. Note that 115 classes, of these 143 classes, consist of a single representative, while the other 28 classes contain two and more representatives. These 28 classes are listed in Appendix A.1. Class members are represented by the numbers of transition functions. Classes are sorted by the minimal number of the member’s transition function. This number of classes will be reduced in further discussion to account other kinds of similarities between structures and to distinguish those of them, which are really different.

By analogy with the classification of the Boolean functions [90], for instance, the PN-classification (stands for “permutation” and “negation”), we call the above partitioning into classes as E-classification (stands for “equality”).

**The “Sierpinsky Gasket” Type of Behavior**

Automaton number 90 generates a self-reproducing structure called the “Sierpinsky gasket”. Figure 3.3 presents the structure generated by this automaton after 64 steps.

![Figure 3.3: Structure, generated by function 90 (Sierpinsky gasket) after 64 steps.](image)

Automaton number 18 generates exactly the same structure as automaton number 90, since the rightmost columns of their transition functions differ only in the rows where combinations in the left parts involve two units. However, this cannot occur for the initial conditions mentioned above. Thus, in the case considered, memoryless automata and automata with memory exhibit the same behavior. It should be noted here that the formula that implements the transition function number 90 seems to be the simplest mathematical description of the Sierpinsky gasket [88, 89, 91].

Note also that the whole class of Sierpinsky gaskets contains eight automata with numbers \(\{18, 26, 82, 90, 146, 154, 210, 218\}\) = \(\mathcal{C}\).

It should be observed that this class does not contain automaton number 22. This automaton is unique in its class and generates a modification of the Sierpinsky gasket that consists of zero and unit triangular substructures. In the structure presented below, adjacent triangular substructures are differently colored.

Replacing zero triangular substructures with zeros and unit triangular substructures with units, we obtain the Sierpinsky gasket mentioned above. Figure 3.5 shows the structure generated by automaton number 22 after 64 steps.
which has the form shown in Fig. 3.8. Rearranging this triangle as shown in Fig. 3.9 can be generated by automaton 60. There are different ways to construct the Pascal triangle [88,92].

3.1 Classification of Structures Generated by 1D Binary Cellular Automata from a Single Seed

Another unique automaton is number 150. It generates self-similar structure, presented in Fig. 3.6. Fig. 3.7 demonstrates the same structure after 64 steps.

The “Binary Pascal Triangle” Type of Behavior

There are different ways to construct the Pascal triangle [88,92]. A binary version can be generated by automaton 60.

With a “binary Pascal triangle”, we mean a triangle of values taken modulo 2, which has the form shown in Fig. 3.8. Rearranging this triangle as shown in Fig. 3.9 we get exactly the structure, generated by automaton 60 (Fig. 3.10).

Automaton number 60 is unique in its invariance class.
Chapter 3. Exploring Evolving 1D Structures and 2D Universal Data Indexing

0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 0 1 0 1 0 0 0 0 0
0 0 0 0 1 1 0 1 0 1 1 0 0 0 0
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
0 0 1 1 1 0 1 1 1 0 1 1 1 0 0
0 1 0 1 0 0 0 1 0 0 1 0 1 0
1 1 0 1 1 0 1 1 1 0 1 1 0 1 1

Figure 3.6: Structure, generated by function 150 in textual form after 8 steps.

Figure 3.7: Structure, generated by function 150 after 64 steps.

1
1 1
1 0 1
1 1 1 1
1 0 0 0 1
1 1 0 0 1 1
1 0 1 0 1 0 1
1 1 1 1 1 1 1

Figure 3.8: The very top of the binary Pascal triangle.

3.1.5 Invariance with Respect to the Operations “Equality” and “Inverse”

As an example, let us consider the inverse of the structure generated by automaton number 90. It is generated by automaton number 165 (Fig. 3.11). Note that in this work we consider inversion up to the zeroth step, because the initial conditions are the same for all the structures, discussed here.

The above interrelation between “direct” and “inverse” structures can serve as a basis for classifying them with respect to the invariance operation “inverse”. However, it is unreasonable to classify with respect to the single operation “inverse”, because each class would contain at most two items and many items would belong to more
3.1. Classification of Structures Generated by 1D Binary Cellular Automata from a Single Seed

Figure 3.9: The very top of the rearranged binary Pascal triangle or the structure, generated by function 90 in textual form after 8 steps.

```
1
1 1
1 0 1
1 1 1 1
1 0 0 0 1
1 1 0 0 1 1
1 0 1 0 1 0 1
1 1 1 1 1 1 1 1
```

Figure 3.10: Structure, generated by function 60 (rearranged binary Pascal triangle) after 64 steps.

Figure 3.11: Structure, generated by function 165 (inverted Sierpinsky gasket) after 64 steps.

than one class. Therefore, the total number of classes will increase significantly and become non-informative.

We suggest that each class assembles structures that are invariant with respect to both operations: “equality” and “inverse”. With these operations being used, for instance, there is a class that assembles nine automata with the numbers $C \cup \{165\} = \{18, 26, 82, 90, 146, 154, 165, 210, 218\}$ (the value of the set $C$ has been defined in Sect. 3.1.4).

It is noteworthy that this example is special, because both the generated structures
and the transition functions are inverse. As was mentioned above this is evidenced by the fact that the sum of their numbers equals 255 (so the rightmost columns of transitions tables for automata 90 and 165 are exactly the inverse of each other).

At the same time the inverses of functions number 18, 26, 82, 146, 154, 210, and 218 do not generate inverse structures for class $C$. In particular, the behavior generated by function number 37 (the inverse of function number 218) is of little interest (see Fig. 3.12).

![Structure, generated by function 37 after 64 steps.](image)

In conclusion, note that there are 135 invariance classes with respect to the operations “equality” and “inverse”. We call this EI-classification.

### 3.1.6 Invariance with Respect to the Operations “Equality” and “Mirror Reflection”

Let us give an example of a structure that is the mirror reflection (with respect to the vertical axis) of the structure generated by automaton number 60. It is generated by automaton number 102 (Fig. 3.13).

![Structure, generated by function 102 (reflected rearranged binary Pascal triangle) after 64 steps.](image)

There are 89 invariance classes with respect to operations “equality” and “mirror reflection”. This classification is called the EM-classification.

Consider the EM-class with the maximum number of representatives: \{2, 10, 16, 24, 34, 42, 48, 56, 66, 74, 80, 88, 98, 106, 112, 120, 130, 138, 144, 152, 162, 170, 176,
184, 194, 202, 208, 216, 226, 234, 240, 248}. This class contains 32 elements, because it unites two E-classes:

- \{2, 10, 34, 42, 66, 74, 98, 106, 130, 138, 162, 170, 194, 202, 226, 234\} – the “left diagonal” structure, for which the rightmost column of the transitions table has the form \(0 1 0 ? 0 ? ? |^T\);

- \{16, 24, 48, 56, 80, 88, 112, 120, 144, 152, 176, 184, 208, 216, 240, 248\} – the “right diagonal” structure, for which the rightmost column of the transitions table has the form \(0 0 0 ? 1 ? ? |^T\).

### 3.1.7 Invariance with Respect to the Operations “Equality”, “Inverse”, and “Mirror Reflection”

Consider the set of four structures \{60, 102, 153, 195\} belonging to the same class with respect to the operations under discussion (see Fig. 3.14).

![Figure 3.14: Structures, generated by functions 60 (upper left), 102 (upper right), 153 (lower left), and 195 (lower right) after 64 steps each.](image)

It can be seen in this figure that structures 60 and 102 are mirror twins, as well as structures 153 and 195. At the same time, structures 60 and 195 are inverses of each other as well as structures 102 and 153.

There are 83 invariance classes with respect to operations “equality”, “inverse”, and “mirror reflection”. This classification is called the EIM-classification. It contains the smallest number of representatives among the classifications considered above.
3.1.8 Invariance with Respect to the Operations “Equality” and “Inverse-Mirror Reflection”

Consider the same set of structures \{60, 102, 153, 195\}. Under the classification with respect to the operations considered, these four automata belong to two different classes: automata with numbers \{60, 153\} form one class and the pair \{102, 195\} forms another one. We called this classification E(I+M)-classification.

There are 135 invariance classes with respect to the operations “equality” and “inverse-mirror reflection”. This coincides with the number of EI-classes. In fact, these two classifications generate similar classes – only 40 structures are differently classified. They fall into four different classes under EI- and E(I+M)-classifications. The differences between these four classes are demonstrated in Table 3.1.

<table>
<thead>
<tr>
<th>EI-classification</th>
<th>E(I+M)-classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>{16, 24, 48, 56, 80, 88, 112, 120, 144, 152, 176, 184, 208, 216, 229, 231, 240, 248}</td>
<td>{16, 24, 48, 56, 80, 88, 112, 120, 144, 152, 173, 176, 184, 189, 208, 216, 240, 248}</td>
</tr>
<tr>
<td>{60, 195}</td>
<td>{60, 153}</td>
</tr>
<tr>
<td>{102, 153}</td>
<td>{102, 195}</td>
</tr>
</tbody>
</table>

Table 3.1: Four classes, which represent the only difference between the EI- and E(I+M)-classifications.

3.1.9 Classification with a Single-Cell Offset

Let us introduce new invariance operations: “offset equality”, “offset inverse”, “offset mirror reflection”, and “offset inverse-mirror reflection”. These operations mean that two structures are examined to mutually correspond either directly or with a vertical, horizontal, or diagonal single-cell offset. As in the case of the “inverse” operation, structures are compared starting from the first timestep, while the initial state is ignored.

Classifications with respect to offset operations will be abbreviated similarly to classifications with respect to direct operations, but with the additional letter “O”.

As an example, consider structures 20 and 155 (see Fig. 3.15, which are equivalent with respect to operation “offset inverse-mirror reflection”. Structure 155 is obtained from structure 20 by means of mirror reflection, inversion, and offset down by a single cell.

There is no point in considering invariance operations with offset by more than a single cell, because the purpose of this paper is to classify together automata with similar behavior, while, in one step, the seed can affect only the cells that are on at most one cell distance from it.

Offset classifications allows reducing the number of classes as compared to classifications without offset. For instance, EIM-classification distinguishes 83 classes,
whereas EIMO-classification reduces their number to 56. These classes are listed in Appendix A.2.

The numbers of classes under classifications with respect to offset invariance operations are presented in column “O$\epsilon$0” of Table 3.3. All the aforementioned in Sect. 3.1.4–3.1.8 invariance operations account no offsets. Column “$\epsilon$0” Table 3.2 contains all the numbers of classes under these classifications.

### 3.1.10 Classification with Errors

It is reasonable to test the structures for correspondence with respect to invariance operations not exactly, but up to several errors, because few mismatches on the first lines of two structures is not a reason for classifying them as unrelated.

As an example, consider structures number 233 and 235 (see Fig. 3.16), which differ only in five cells.

![Figure 3.16: Structures, generated by functions 233 (left) and 235 (right) after 16 steps each. Here the structures are four times magnified with respect with the majority of figures above in this section.](image)

Classifications with respect to operations that account for errors will be abbreviated similarly to classifications with respect to exact operations, but with the addition of “$\epsilon n$”, where $n$ is the maximal number of mismatches.

For $n > m$ then $\epsilon n$-classification contains less classes then the $\epsilon m$-classification. Obviously, they will be quite similar, because each of the $\epsilon n$-classes will represent one

---

$^\dagger$The meaning of the suffix “$\epsilon 0$” is explained in Sect. 3.1.10.
or more $\epsilon m$-classes.

Columns “$\epsilon 0$”–“$\epsilon 6$” of Tables 3.2 and 3.3 contain the numbers of classes in various classifications with zero to six mismatches. Note that, in the case of no offset (Table 3.2), all the values in rows $EI$ and $E(I+M)$ coincide, whereas the corresponding values in Table 3.3 are different.

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon 0$</th>
<th>$\epsilon 1$</th>
<th>$\epsilon 2$</th>
<th>$\epsilon 3$</th>
<th>$\epsilon 4$</th>
<th>$\epsilon 5$</th>
<th>$\epsilon 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E$</td>
<td>143</td>
<td>134</td>
<td>129</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>127</td>
</tr>
<tr>
<td>$EM$</td>
<td>89</td>
<td>83</td>
<td>81</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>79</td>
</tr>
<tr>
<td>$EI$</td>
<td>135</td>
<td>124</td>
<td>120</td>
<td>119</td>
<td>119</td>
<td>119</td>
<td>118</td>
</tr>
<tr>
<td>$E(I+M)$</td>
<td>135</td>
<td>124</td>
<td>120</td>
<td>119</td>
<td>119</td>
<td>119</td>
<td>118</td>
</tr>
<tr>
<td>$EIM$</td>
<td>83</td>
<td>76</td>
<td>74</td>
<td>73</td>
<td>73</td>
<td>73</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 3.2: Number of classes in various classifications without offsets.

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon 0$</th>
<th>$\epsilon 1$</th>
<th>$\epsilon 2$</th>
<th>$\epsilon 3$</th>
<th>$\epsilon 4$</th>
<th>$\epsilon 5$</th>
<th>$\epsilon 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$EO$</td>
<td>125</td>
<td>118</td>
<td>118</td>
<td>117</td>
<td>117</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>$EMO$</td>
<td>78</td>
<td>75</td>
<td>75</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td>74</td>
</tr>
<tr>
<td>$EIO$</td>
<td>88</td>
<td>86</td>
<td>86</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>$E(I+M)O$</td>
<td>88</td>
<td>85</td>
<td>85</td>
<td>84</td>
<td>84</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>$EIMO$</td>
<td>56</td>
<td>55</td>
<td>55</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td>54</td>
</tr>
</tbody>
</table>

Table 3.3: Number of classes in various classifications with offsets.

### 3.1.11 Discussion

In this section, seventy classifications of structures generated by one-dimensional cellular automata from a single seed are presented. The numbers of classes, according to all variants of classifications, are presented in Tables 3.2 and 3.3. EIMO-classifications family brings the most informative structures grouping. All 256 structures can be subdivided into 54 different kinds of shapes. It is also demonstrated that there are automata with cells which have memory and automata with memoryless cells exhibiting the same behavior.

Reference [50] contains supplementary material for this work.
3.2 Generalized Coordinates for Cellular Automata Grids

The topic of this section will be two-dimensional grids of cellular automata (the space $G$ from its formal definition in Sect. 1.3) composed of regular polygons and non-standard metrics, which can be defined on them. We are going to introduce a way of mapping a two-dimensional grid to a one-dimensional structure, providing a concept of universal data organization. At the same time, the approach should not be based on the adjacency lists caching in memory or files. It should provide a methodology to quickly find neighbours. The benefits and disadvantages of this approach are discussed in Sect. 3.3.

There are only three two-dimensional grids of cellular automata composed of regular polygons: a grid of triangles (Fig. 3.17), a grid of squares (Fig. 3.18) and a grid of hexagons (Fig. 3.19).

![Figure 3.17: Two-dimensional grid of triangles with Cartesian coordinates introduced. The nearest neighbours of the origin of the coordinates are shown with grey. Main neighbours are darker.](image)

The nearest neighbours of a cell are cells, which share at least one common vertex\(^\ddagger\). In Figs. 3.17–3.19 the origin of the coordinates is outlined and its nearest (or immediate) neighbours are shown in grey on the figures. The cells, which have a common edge (or two vertexes) are called “main neighbours”. They are shown in darker grey.

A neighbourhood $N$, consisting of main neighbours only, is called a “von Neumann neighbourhood”. A neighbourhood containing all nearest neighbours is called a “Moore

---

\(^\ddagger\)In the figures in this section cells are moved apart for the visual representation reasons and to be able to outline cells, when needed. It is still obvious, which cells have common vertex or edge.
Figure 3.18: Two-dimensional grid of squares with Cartesian coordinates introduced. The nearest neighbours of the origin of the coordinates are shown with grey. Main neighbours are darker.

Figure 3.19: Two-dimensional grid of hexagons with Cartesian coordinates introduced. All the nearest neighbours of the origin of the coordinates are the main neighbours of it. They are shown in dark grey.

neighbourhood”.

3.2.1 Basic Concepts

A key notion is that “an $i$-th ring” of a distinct cell. In the sequel of this section “neighbours” refers to nearest neighbours of the cell (in other words, we will use the Moore neighbourhood). Define neighbours of a cell to be the set of cells of the “first ring”. The definition of the $i$-th ring is constructed recursively: for a cell $a$, the set of cells of the $i$-th ring is a union of all members of the first rings for all cells, which belong to the $(i - 1)$-th ring of cell $a$, excluding all the cells, which are members of the lower order rings of cell $a$. 
Formally, if \( R(a, i) \) is a set of cells of the \( i \)-th ring of a cell \( a \), then one may write.

\[
R(a, i) = \{ b \mid \exists c : b \in R(c, 1); c \in R(a, i - 1); \not\exists j < i : b \in R(a, j) \}. \quad (3.7)
\]

Figures 3.20–3.22 show the sequence of the first four rings for the origin of coordinates for all two-dimensional grids, composed of regular hexagons.

![Diagram of cellular automata grid](image)

Figure 3.20: First four rings for the origin of coordinates for the two-dimensional grid of triangles. Adjacent rings are shown with different colors.

On the bigger rings one can distinguish “corners”. Along the edges of the rings all cells has the same pattern of neighbours, which belong to the same ring. Corner cells can be distinguished as cells, where this pattern changes. For example, if we consider the forth ring of the origin of coordinates for the grid of hexagons, shown in Fig. 3.22, then cells \((-1; 4), (0; 4), \) and \((1; 4)\) belong to the upper edge of the ring, when \((-4; 4)\) and \((4; 4)\) are corners.

It is useful to use assume that

\[
a = R(a, 0), \quad (3.8)
\]

the zero-ring of a cell is the cell itself.

Now we may say that for any cells \( a \) and \( b \) there is one and only one index \( i \), such that \( b \) is in \( R(a, i) \).

Moreover, an important property of the ring concept is that if \( a \) is in \( R(b, i) \) then \( b \) is in \( R(a, i) \).
Figure 3.21: First four rings for the origin of coordinates for the two-dimensional grid of squares. Adjacent rings are shown with different colors.

The notion of the ring may be easily generalized for different types of cells’ neighbourhoods and also for grids of larger dimensions.

When the problem requires an $n$-dimensional space, then the grid $G$ usually represents a discrete space of the same dimensionality. Each cell can be defined with the help of $n$-dimensional coordinates (Cartesian in most cases). Usage of generalized coordinates is the way to represent an $n$-dimensional discrete space by a one-dimensional sequence.

The idea is to enumerate all cells of the grid with non-negative integer numbers, which we will call generalized coordinates. They should be used in a row from zero to the amount of cells minus one. Each number is to have one and only one corresponding cell.

The choice of a cell, which gets zero coordinate, is not a subject of a discussion, because the system is invariant relative to its position.

To be able to execute a transition function over the grid after the introduction of such enumeration one need to provide a mechanism of nearest neighbours foundation for any given cell. In case when the neighbourhood $N$ contains other than only nearest neighbours, generalized coordinates would be applicable too, because then neighbouring cells could be found recursively, as the nearest neighbours of the nearest neighbours and so on.
Several methods to associate cells with generalized coordinates exist. The main point is to introduce them, providing an ability to get neighbours for any cell and do it fast. In Sect. 3.2.2 we will consider spiral coordinates introduction. In Sect. 3.2.3 an effective composite method will be presented for the grid of triangles. This method is based on the spiral generalized coordinates for the grid of hexagons.

### 3.2.2 Spiral Generalized Coordinates

When using the spiral approach for generalized coordinates, the algorithm for cell enumeration is very straightforward: one cell is chosen as the origin and is associated with number zero. All other cells are enumerated from one ring of the zeroth cell to the other, for example, clockwise in each ring, starting from the “upper-right corner” of the ring\(^5\). Consider all possible two-dimensional grids, composed of the regular polygons.

\(^5\)What is the “upper-right corner” of the ring will be shown for each type of the grid. This should become obvious from the illustrations, which accompany this section.
The Grid of Triangles

Spiral generalized coordinates for a grid of triangles is presented in Fig. 3.23. As was done above in Fig. 3.20–3.20, adjacent rings are shown in different colors: white or light grey. At the same time the corner cells of the rings are shown with dark grey. Corners of the rings are also associated with six indices.

To provide a mechanism to determine a cell’s nearest neighbour enumerate cells of the first ring with indices from 0 to 11 as it is shown in Fig. 3.24. One of the specific properties for the grid of triangles is that two possible orientations for the cells should be distinguished. These orientations will be designated as “▲” and “▼” in further discussion.

For the cell in the ▲-orientation (Fig. 3.24, left), enumeration of neighbours first runs over the main neighbours, and after this it restarts from the upper-left corner. At the same time, the indexation for the ▼-orientation is obtained from the first one by the rotation. This provides an invariance in terms of the orientation, that will be exploited below.

A set of functions should be defined for further use. First consider the length of the \( i \)-th ring’s edge. Usually, it will be denoted as \( l_i \), when all edges of the rings are
equal, as they are for grids of squares and hexagons. However, for a grid of triangles in our variant of enumeration, according to mnemonic rule, mentioned in the beginning of the section, edges have different lengths. The \(i\)-th ring for the case represents an irregular hexagon of \(6i\) cells with edges of three different lengths. The length of an edge is defined as \(l_{i,j}\), where \(j\) is an index of the corner, from which the edge starts, being drawn clockwise (so, \(j\) can take values from 0 to 5).

For the first ring is can be observed that

\[
\begin{align*}
    l_{1,0} &= 3 \\
    l_{1,1} &= 3 \\
    l_{1,2} &= 2 \\
    l_{1,3} &= 5 \\
    l_{1,4} &= 2 \\
    l_{1,5} &= 3.
\end{align*}
\] (3.9)

The length of higher order rings can be recursively computed as in Eqs. 3.10.

\[
\begin{align*}
    l_{i,0} &= 2i + 1 \\
    l_{i,1} &= 2i + 1 \\
    l_{i,2} &= 2i \\
    l_{i,3} &= 2i + 3 \\
    l_{i,4} &= 2i \\
    l_{i,5} &= 2i + 1.
\end{align*}
\] (3.10)

These formulae are not correct for the zeroth cell. Later we will consider the origin of coordinates separately.

In Fig. 3.23 corners’ indices are shown outside the grid fragment. Denote the coordinate of a \(j\)-th corner of an \(i\)-th ring as \(C_{i,j}\).

Eqs. 3.10 allow to claim that the perimeter of each next ring is twelve cells longer than for the previous one. Knowing that the perimeter of the first ring is twelve cells, it is possible to conclude that

\[
C_{i,0} = 12 \cdot \Sigma(i - 1) + 1, \quad (3.11)
\]
Chapter 3. Exploring Evolving 1D Structures and 2D Universal Data Indexing

where

\[ \Sigma(i) = 1 + 2 + \ldots + (i - 1) + i = \sum_{k=1}^{i} k = \frac{i^2 + i}{2}. \quad (3.12) \]

Each corner can now be calculated, knowing the previous one and the length of the edge between them. The difference between the \( j-1 \)-th and \( j \)-th corners’ of the same ring \( i \) equals to \( l_{i,j-1} - 1 \). So, for the triangular grid, each corner of any ring can be found with the help of Eq. 3.13.

\[ C_{i,j} = 12 \cdot \Sigma(i-1) + 1 + \sum_{k=0}^{j-1} (l_{i,k} - 1) = 6(i^2 - i) + 1 + 2i \cdot j + \begin{cases} -1, & \text{if } j = 3 \\ 1, & \text{if } j = 4 \\ 0, & \text{otherwise} \end{cases} \quad (3.13) \]

The number of the ring, which contains a cell \( a \) will be denoted as \( n(a) \) and can be computed as follows:

\[ n(a) = \left\lfloor \sqrt{1 + 8 \cdot \lceil a/12 \rceil} - 1 \right\rfloor. \quad (3.14) \]

Let’s denote the \( i \)-th neighbour of the cell \( a \) as \( N_i(a) \) (\( i \) is from 0 to 11 as shown in Fig. 3.24).

It will be sufficient to introduce a method, which allows to find only three main neighbours, because all other neighbours can be generated via:

\[
\begin{align*}
N_3(a) &= N_2(N_0(a)) \\
N_4(a) &= N_1(N_2(N_0(a))) = N_0(N_2(N_1(a))) \\
N_5(a) &= N_2(N_1(a)) \\
N_6(a) &= N_0(N_1(a)) \\
N_7(a) &= N_2(N_0(N_1(a))) = N_1(N_0(N_2(a))) \\
N_8(a) &= N_0(N_2(a)) \\
N_9(a) &= N_1(N_2(a)) \\
N_{10}(a) &= N_0(N_1(N_2(a))) = N_2(N_1(N_0(a))) \\
N_{11}(a) &= N_1(N_0(a)).
\end{align*}
\]

Determining the neighbours for the cell \( a \) starts with the calculation of the ring number to which it belongs, using Eq. 3.14. Then one may determine the position of cell \( a \) inside the ring, comparing its coordinates with corner coordinates calculated using Eq. 3.13. All 13 possible positions (it could be any corner or on any edge of the ring) have a corresponding row in Table 3.4. On the edges the two possible cell orientations should be also distinguished, when orientation of corner cells is known beforehand.

\footnote{Here and further we also distinguish a last cell in the ring \( C_{n(a) + 1,0} - 1 \), which is not a corner, but also belongs to a special cases. Saying “corner” in context of the position of the cell inside the ring, we will roughly mean this individual case too.}
In addition, the origin of coordinates requires a separate discussion, because it represents all corners of the zero ring simultaneously. Following definitions should be added: $N_0(0) = 12$, $N_1(0) = 4$, $N_2(0) = 8$, $N_0(12) = 0$, $N_1(4) = 0$, and $N_2(8) = 0$.

Presented approach is using recursive calls to determine non-main neighbors of a

---

### Table 3.4: Formulae for calculating all the main neighbours of cell $a$

<table>
<thead>
<tr>
<th>Position</th>
<th>▲/▼</th>
<th>Neighbour’s Index</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$C_{n(a),0}$</td>
<td>▲</td>
<td>$C_{n(a)+2,0} + \cdot 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$a - 1$</td>
</tr>
<tr>
<td>$(C_{n(a),0}; C_{n(a),1})$</td>
<td>▲</td>
<td>$a - 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$a + 1$</td>
</tr>
<tr>
<td>$C_{n(a),1}$</td>
<td>▲</td>
<td>$a - 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$a + 1$</td>
</tr>
<tr>
<td>$(C_{n(a),1}; C_{n(a),2})$</td>
<td>▲</td>
<td>$a - 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$a + 1$</td>
</tr>
<tr>
<td>$C_{n(a),2}$</td>
<td>▲</td>
<td>$C_{n(a)+1,3} - 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$C_{n(a)-1,3} + a - C_{n(a),3} - 1$</td>
</tr>
<tr>
<td>$(C_{n(a),3}; C_{n(a),4})$</td>
<td>▲</td>
<td>$a + 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$a - 1$</td>
</tr>
<tr>
<td>$C_{n(a),3}$</td>
<td>▲</td>
<td>$a - 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$a + 1$</td>
</tr>
<tr>
<td>$(C_{n(a),4}; C_{n(a),5})$</td>
<td>▲</td>
<td>$a + 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$a - 1$</td>
</tr>
<tr>
<td>$C_{n(a),5}$</td>
<td>▲</td>
<td>$C_{n(a)+1,5} + 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$C_{n(a)-1,5} + a - C_{n(a),5} - 1$</td>
</tr>
<tr>
<td>$(C_{n(a),0}; C_{n(a),1})$</td>
<td>▲</td>
<td>$C_{n(a)+1,5} + a - C_{n(a),5} - 1$</td>
</tr>
<tr>
<td></td>
<td>▼</td>
<td>$C_{n(a)-1,5} + a - C_{n(a),5} - 1$</td>
</tr>
</tbody>
</table>

Table 3.4: Formulae for calculating all the main neighbours of cell $a$, depending on its position inside the ring $n(a)$ (can be calculated for the given $a$ with the help of the Eq. 3.14) and orientation. All possible doublets of positions and orientations are presented in first two columns. The remaining columns correspond to each distinct neighbour’s index, as in Fig. 3.24.
cell. This has negative consequences to the productivity of computations, but, in the same time, if the recursion will be eliminated then the amount of positions, which should be considered, will rise dramatically.

The difference between two cases of cell’s orientation on the edges is following:

- signs in $a \pm 1$ members are changed to the opposite;
- sign in $C_{n\pm1,k} + a - C_{n,k} - 1$ member is changed to the opposite.

**The Grid of Squares**

The following discussion will follow the same sequence of actions as for the grid of triangles. A possible spiral generalized coordinates the grid of squares is presented in Fig. 3.25. Corners of the rings are associated with four indices.

![Figure 3.25: Spiral generalized coordinates for the grid of squares. Adjacent rings are shown in different colors: white or light grey. All rings’ corners are colored in dark grey. Corners’ indices are shown outside of the grid’s fragment.](image)

To determine the nearest neighbour of a cell enumerate the cells of the first ring with indices from 0 to 7 as shown in Fig. 3.26.

The same set of the functions discussed above should be defined. The length of the $i$-th ring edge is determined by
3.2. Generalized Coordinates for Cellular Automata Grids

Figure 3.26: Neighbours’ indices for the basic neighbourhood for the grid of squares. Main neighbours are darkened.

\[ l_i = 2i + 1. \]  \hspace{1cm} (3.16)

The coordinate of the \( j \)-th corner of the \( i \)-th ring (\( j \) is from 0 to 3) is \( C_{i,j} \). For zero corners we find

\[ C_{i,0} = l_{i-1}^2 = (2i - 1)^2. \]  \hspace{1cm} (3.17)

Each next corner lies \( l_i - 1 \) cells from the previous one. Each corner of any ring can be found by

\[ C_{i,j} = l_{i-1}^2 + j \cdot (l_i - 1) = (2i - 1)^2 + i \cdot j. \]  \hspace{1cm} (3.18)

The number of the ring, which contains a cell \( a \), can be found by

\[ n(a) = \left\lceil \frac{\sqrt{a+1}}{2} \right\rceil. \]  \hspace{1cm} (3.19)

It will be sufficient to find only the four main neighbours, because

\[ N_4(a) = N_0(N_1(a)) = N_1(N_0(a)) \]
\[ N_5(a) = N_1(N_2(a)) = N_2(N_1(a)) \]
\[ N_6(a) = N_2(N_3(a)) = N_3(N_2(a)) \]
\[ N_7(a) = N_3(N_0(a)) = N_0(N_3(a)). \]  \hspace{1cm} (3.20)

Determining the neighbours for the cell \( a \) starts from the calculation of the ring number, to which it belongs, using Eq. 3.19. Then one may determine the position of the cell \( a \) inside the ring, comparing its coordinates with the corner’s coordinates calculated using Eq. 3.18. All 9 possible positions have a corresponding row in Table 3.5.

Again, the origin of coordinates requires a separate discussion, because it represents all corners of the zeroth ring simultaneously. The following definitions should be added: \( N_0(0) = 2, N_1(0) = 4, N_2(0) = 6, N_3(0) = 8, N_2(2) = 0, N_3(4) = 0, N_0(6) = 0, \) and \( N_1(8) = 0. \)
The number of the ring, which contains a cell \( n(a) \), can be found with

\[
n(a) = \left\lfloor \frac{\sqrt{1 + 8 \cdot [a/6] - 1}}{2} \right\rfloor.
\] (3.24)
3.2. Generalized Coordinates for Cellular Automata Grids

Figure 3.27: Spiral generalized coordinates for the grid of hexagons. Adjacent rings are shown in different colors: white or light grey. All rings’ corners are colored in dark grey. Corners’ indices are shown outside of the grid’s fragment.

Figure 3.28: Neighbours’ indices for the basic neighbourhood for the grid of hexagons. Main neighbours are darkened.

All cell neighbours are main on the grid of hexagons. Nevertheless, it will be enough to formulate a method, which allows to find only four by

\[ N_0(a) = N_5(N_1(a)) = N_1(N_5(a)) \]
\[ N_3(a) = N_4(N_2(a)) = N_2(N_4(a)). \]

Despite of this fact, we will show how to find all neighbours without recursive
operations.

Determining of neighbours for cell $a$ starts from the calculation of the ring number using Eq. 3.24. Then one may determine the position of the cell $a$ inside the ring, comparing its coordinates with corner’s coordinates calculated using Eq. 3.23. All 13 possible positions have a corresponding row in Table 3.6.

The origin requires a separate discussion, because it represents all corners of the zeroth ring simultaneously. The following definitions should be added:

\[ N_0(0) = 1, \]
\[ N_1(0) = 2, N_2(0) = 3, N_3(0) = 4, N_4(0) = 5, N_5(0) = 6, N_3(1) = 0, N_4(2) = 0, N_5(3) = 0, N_0(4) = 0, N_1(5) = 0, \text{and } N_2(6) = 0. \]

### 3.2.3 Composite Generalized Coordinates for the Grid of Triangles

An effective composite method for generalized coordinates for a grid of triangles will be discussed. It is based on the spiral coordinates for a grid of hexagons, described in Sect. 3.2.2.

We use the fact that six triangles form a hexagon as in Fig. 3.29. So, we may distinguish a “metagrid” of hexagons over the triangular grid. Generalized coordinates may be introduced in the following way: first, spiral generalized coordinates should be applied to such hexagons. Next a coordinate of a triangle can be retrieved as a coordinate of a hexagon, multiplied by 6 with the added index of triangle inside the hexagon. These indices can be seen in Fig. 3.29 in the zeroth hexagon.

In other words, if $a$ is a cell on the grid of triangles (and, in the same time, it is an integer number, which is equal to its generalized coordinate), $H(a)$ is a spiral coordinate of composite hexagon, which holds it, and $h(a)$ is an index of the corresponding triangle within the hexagon.

\[ a = 6H(a) + h(a). \]  
\[ H(a) = \lfloor a/6 \rfloor \]  
\[ h(a) = a \mod 6, \]  

The resulting enumeration is presented in Fig. 3.29. Note that grid of hexagons is turned counterclockwise in comparison with Fig. 3.27.

The same neighbour enumeration, as shown in Fig. 3.24 will be used for this method. Coordinates of the $m$-th neighbours of a cell $a$ can be retrieved, using the following algorithm.

1. Calculate $H(a)$ and $h(a)$, using Eqs. 3.27.

2. Having $h(a)$ and $m$ get the corresponding value from Table 3.7 (lets denote it as $M$). If there is a diamond ($\diamond$) instead of a number in the corresponding box of the table, then the $m$-th neighbour of the cell $a$ on the hexagonal metagrid belongs to the same hexagon $H(a)$ as the cell $a$ does. Otherwise it belongs to the $M$-th neighbour of the hexagon $H(a)$ in terms of spiral coordinates for the grid of hexagons. That is why $M$ is from 0 to 5, and moreover it may be equal only to $h(a)$, $h(a) + 1$ or $h(a) + 2$ modulo 6. So, each row contains exactly
### Table 3.6: Formulae for calculating all the main neighbours of the cell \( a \), depending on its position inside the ring \( n(a) \) (can be calculated for the given \( a \) with the help of the Eq. 3.24). All possible positions are presented in the first column. The rest columns correspond to each distinct neighbour’s index, adduced in Fig. 3.28.

<table>
<thead>
<tr>
<th>Position</th>
<th>Neighbour’s Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{n(a),0} )</td>
<td>( C_{n(a)+1,0} )</td>
<td>( C_{n(a)+1,0}+1 )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,0} )</td>
<td>( C_{n(a)+1,0}-1 )</td>
<td>( C_{n(a)+2,0}-1 )</td>
<td></td>
</tr>
<tr>
<td>( (C_{n(a),0};C_{n(a),1}) )</td>
<td>( a-C_{n(a),0} )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,0} )</td>
<td>( a-C_{n(a),0}-1 )</td>
<td>( a+1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{n(a),1} )</td>
<td>( C_{n(a)+1,1}-1 )</td>
<td>( C_{n(a)+1,1} )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,1} )</td>
<td>( a-1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (C_{n(a),1};C_{n(a),2}) )</td>
<td>( a-1 )</td>
<td>( C_{n(a)+1,1} )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,1} )</td>
<td>( a-C_{n(a),1} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{n(a),2} )</td>
<td>( a+1 )</td>
<td>( C_{n(a)+1,2} )</td>
<td>( a-1 )</td>
<td>( C_{n(a)+2,0}-1 )</td>
<td>( a+C_{n(a),1} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (C_{n(a),2};C_{n(a),3}) )</td>
<td>( a-C_{n(a),2}-1 )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,2} )</td>
<td>( a-C_{n(a),2} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{n(a),3} )</td>
<td>( C_{n(a)-1,3} )</td>
<td>( a+1 )</td>
<td>( C_{n(a)+1,3} )</td>
<td>( a-C_{n(a),1} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (C_{n(a),3};C_{n(a),4}) )</td>
<td>( a-C_{n(a),3} )</td>
<td>( a-C_{n(a),3}-1 )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,3} )</td>
<td>( a-C_{n(a),3}-1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{n(a),4} )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,4} )</td>
<td>( a+1 )</td>
<td>( C_{n(a)+1,4} )</td>
<td>( a-C_{n(a),4} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (C_{n(a),4};C_{n(a),5}) )</td>
<td>( a-C_{n(a),4} )</td>
<td>( a-C_{n(a),4}-1 )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,4} )</td>
<td>( a-C_{n(a),4}-1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{n(a),5} )</td>
<td>( C_{n(a)+1,5}+1 )</td>
<td>( a+1 )</td>
<td>( C_{n(a)+1,5} )</td>
<td>( a-1 )</td>
<td>( C_{n(a)+1,5} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( (C_{n(a),5};C_{n(a)+1,0}-1) )</td>
<td>( a-C_{n(a),5} )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,5} )</td>
<td>( a-C_{n(a),5}+1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{n(a)+1,0}-1 )</td>
<td>( C_{n(a)+1,0}+1 )</td>
<td>( a+1 )</td>
<td>( C_{n(a)-1,0} )</td>
<td>( a-C_{n(a),1} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2. Generalized Coordinates for Cellular Automata Grids
five diamonds, two cells with \( h(a) \), three cells with \( h(a) + 1 \), and two cells with \( h(a) + 2 \).

3. Knowing, which hexagon the \( m \)-th neighbour of the cell \( a \) belongs to, and having \( h(a) \) and \( m \) find the index of the required cell inside the hexagon (number from 0 to 5) in the Table 3.8.

4. After finding the coordinate of the containing hexagon and the index of the cell inside it, the coordinate of the \( m \)-th neighbour for the cell \( a \) can be calculated, using Eq. 3.27.

The advantage of this approach to the enumeration of neighbours results from the fact that there is no need for two separate tables for different cells’ orientations. All cells with even numbers are ▲-oriented, whereas all cells with odd numbers are ▼-oriented.

Tests show, that this variant of the generalized coordinates for the grid of triangles allows to find the nearest neighbours more than four times faster than the spiral
3.2. Generalized Coordinates for Cellular Automata Grids

Table 3.7: Indices of the neighbouring hexagons (in terms of the metagrid with spiral coordinates introduced), which contain the nearest neighbours for the distinct triangle cells. If “◦” is found in the corresponding cell, then the required neighbour belongs to the same hexagon $H(a)$, which the cell $a$ belongs to.

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
h(a) & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\hline
0 & 1 & ◦ & ◦ & 1 & 2 & 2 & ◦ & ◦ & ◦ & 0 & 0 & 1 \\
1 & ◦ & ◦ & 2 & ◦ & ◦ & ◦ & 1 & 1 & 2 & 2 & 3 & 3 \\
2 & ◦ & 3 & ◦ & 2 & 2 & 3 & 3 & 4 & 4 & ◦ & ◦ & ◦ \\
3 & 4 & ◦ & ◦ & 4 & 5 & 5 & ◦ & ◦ & ◦ & 3 & 3 & 4 \\
4 & ◦ & ◦ & 5 & ◦ & ◦ & ◦ & 4 & 4 & 5 & 5 & 0 & 0 \\
5 & ◦ & 0 & ◦ & 5 & 5 & 0 & 0 & 1 & 1 & ◦ & ◦ & ◦ \\
\hline
\end{array}
\]

Table 3.8: Indices of the nearest neighbours inside hexagons of the metagrid.

\[
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
h(a) & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\hline
0 & 3 & 1 & 5 & 2 & 5 & 4 & 2 & 3 & 4 & 2 & 1 & 4 \\
1 & 2 & 0 & 4 & 3 & 4 & 5 & 3 & 2 & 5 & 3 & 0 & 5 \\
2 & 1 & 5 & 3 & 4 & 3 & 0 & 4 & 1 & 0 & 4 & 5 & 0 \\
3 & 0 & 4 & 2 & 5 & 2 & 1 & 5 & 0 & 1 & 5 & 4 & 1 \\
4 & 5 & 3 & 1 & 0 & 1 & 2 & 0 & 5 & 2 & 0 & 3 & 2 \\
5 & 4 & 2 & 0 & 1 & 0 & 3 & 1 & 4 & 3 & 1 & 2 & 3 \\
\hline
\end{array}
\]

variant, presented in Sect. 3.2.2. This is caused by the complexity of a ring for the grid of triangles and by deep recursive calls, which are required, when using the spiral coordinates.

One may say that the efficiency of the spiral approach for the grid of triangles can be significantly increased, by providing explicit formulae for the non-main neighbours. Of course, this will eliminate recursion, but the complexity of a ring will aggravate the situation, requiring much more positions to be considered.

Moreover, the composite approach has another important benefit: any neighbourhood for a distinct cell requires close to the same amount time. Nevertheless, when dealing with spiral coordinates, if calculation of neighbours with indices 0, 1, and 2 (see Fig. 3.24) need an average time $t$, then calculation of neighbours 3, 5, 6, 8, 9, and 11 will require an average time $2t$. Finally, finding neighbours 4, 7, and 10 will take an average time $3t$.

The composite approach presents a general idea, which can be used in the different manners: triangles can be grouped into a metagrid of parallelograms (metrics will be the same as for squares), squares can be united into shapes similar to hexagons and so on.
3.3 Discussion

Generalized coordinates provide a universal approach to data storage. For different kind of grids, regardless dimensionality, the same data structure – a one-dimensional chain, is sufficient. The only thing that needs to be changed for a specific cellular automaton is a set of expressions for the cells nearest neighbours calculation. Serial data can be serialized [16] and stored straightforwardly. All stream-based processing algorithms (for example, analysis, preprocessing, compression or other) are immediately applicable.

Grid may be easily enlarged if necessary. Appending cells to the end of a chain is simpler than a reallocation of a multidimensional block structure. At the same time, it can be enlarged in the spiral or another, but a predefined way. This is not suitable for a case when unstructured grids need to be simulated. If the method of coordinates introduction does not allow to follow the structure of a grid, then prebuilding of adjacency lists of coordinates looks advantageous.

Moreover, adjacency lists can be of use to simulate periodic boundary conditions. They can be combined with spiral generalized coordinates to find neighbours on the boundary. At the same time, spiral generalized coordinates allow convenient of constant boundary conditions usage.

Offered approach in general has worse productivity being compared with the Cartesian metrics case. We performed tests on the example of the Game of Life [78] modelling on the $201 \times 201$ grid of squares. As a result we see that when using the generalized coordinates the simulation is less than 1% slower. When the algorithm is more complex the benefit of Cartesian coordinates will become negligible. Nevertheless, the overhead will be more significant, when running experiments on a three-dimensional grid.

We also performed efficiency tests for the methods of generalized coordinates, discussed in Sect. 3.2.2 and 3.2.3. Test were run on five different computing systems. If we will assume the average time required to find a random neighbour using the slowest method (spiral coordinates for the grid of triangles) as unity, then the method of the

- spiral coordinates for the grid of squares is 3.92 times faster;
- composite coordinates for the grid of triangles is 4.44 times faster;
- spiral coordinates for the grid of hexagons is 6.17 times faster.

Cellular automata can be considered as general models of parallel computations with in principle infinite extent of parallelism [17]. The same functionality could be achieved with the help of infinite amount of Turing machines [23, 24]. Using generalized coordinates a system with infinite extent of parallelism can be mapped to the system of finite quantity of Turing machines. A single machine used to represent a data storage (a tape) and finite quantity of machines are used for neighbours calculation and synchronization. Their exact amount depends on the implementation.\[\parallel\]When speaking about spiral generalized coordinates.
Spiral generalized coordinates make a problem for the parallel computations, based on domain decomposition, but there are many other ways of such coordinates introduction, which remain out of the scope of this chapter. When the enumeration of cells will follow the Peano curve [93, 94] (for the grid of squares) or the Gosper-Peano curve (for the grid of hexagons) of other space filling curves [93], it would be possible to decompose the grid efficiently.