Chapter 10

Hard, firm, soft … Etherealware: Computing by Temporal Order of Clocking

Michael Vielhaber

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.80432

Abstract

We define *Etherealware* as the concept of implementing the functionality of an algorithm by means of the clocking scheme of a cellular automaton (CA). We show, which functions can be implemented in this way, and by which CAs.

Keywords: cellular automaton, etherealware, asynchronous, update rule, universality, temporal order, clocking-computable

Your task: Compute a lot of different functions on n-bit inputs.

Your device: A (fixed!) cellular automaton (CA) on the (fixed!) ring or torus topology with n cells, is capable of holding one bit each.

You may not change the CA (its update rule) nor the topology. You may not enter additional information in the form of parameters (there would be no space to store them anyway) — and yet you are supposed to evaluate many different functions. The available degree of freedom is the clocking scheme of the cells, anything from synchronous to completely asynchronous is allowed.

Can you do it?

The perhaps surprising answer is: *yes!*

Every bijective function on the set $\{0, 1, \ldots, 2^n - 1\}$, which acts as an even permutation is clocking-computable, as well as many non-bijective functions.

1. Introduction

1.1. What is etherealware?

Computation takes place in dedicated *hard*ware or on general-purpose *hard*ware by dedicated *soft*ware. Different functionality requires either changing the *hard*ware (think ASIC, FPGA) or changing the *soft*ware running on it. *Firm*ware is an intermediate concept, where the *hard*ware is modified by microprogramming a CPU or personalizing an FPGA.

Etherealware is the first way to use fixed hardware (certain cellular automata (CA) in this case), run fixed software (the same update rule for all cells, for all time, for all purposes), and still deliver diversity in the resulting function: by changing *only* the clocking scheme, the order in which the CA's cells are updated.

1.2. State-of-the-art

The study of synchronous CAs starts with Wolfram [1]. We use asynchronous CAs as deterministic devices with a finite number of computation steps, which is a *new* point of view.

Previously, asynchronous CAs have been treated as dynamical systems, where infinite computations are considered, and the focus lies on concepts like orbits, fixed points, ergodicity, transients, cycles and their periods, and long-term behavior. Also, randomness can be introduced to average over many possible asynchronous schemes. Papers in this respect are:

Ingerson and Buvel [2], 1984, distinguish synchronous, random (completely asynchronous), and periodic clocking, which yield clearly distinguishable behavior.

Barrett et al. [3–6], 1999–2003, consider sequential dynamical systems (SDS), including CAs with arbitrary toplogy and neighborhoods. They cover random graphs as topology and dynamical systems topics such as fixed points and invertibility.

Siwak [7], 2002, gives an overview of simulating machines, including CAs and SDSs, and unifies them under the concept of "filtrons."

Lee *et al*. [8], 2003, give an asynchronous CA on the two-dimensional grid $\mathbb{Z} \times \mathbb{Z}$, which is Turing universal.

Laubenbacher and Pareigis [9], 2006, build upon [3–6] and observe that not all $n!$ permutations of the cells lead to different temporal rules. Their equivalence classes coincide—for our setting, CAs on the torus—with our result ([10], Thm. 1]).

Fatès et al. [11], 2006, consider ECAs with quiescent states ($000 \mapsto 0, 111 \mapsto 1$, i.e., with even Wolfram rule ≥ 128). They show that 9 ECAs diverge, while the other 55 converge to a random fixed point, in 4 clearly distinguishable time frames $\Theta(n \log(n)), \Theta(n^2), \Theta(n^3)$, or $\Theta(n2^n)$ with characteristic behavior per time frame.

Macauley, McCammond, and Mortveit [12, 13], 2007–2010, also treat SDSes, in particular ECAs. For each ECA, [13] gives the periodic states and the dynamics group. Conjecture 5.10 in [13] about Wolfram rule 57 coincides with our finding that ECA-57 generates the alternating groups

http://dx.doi.org/10.5772/intechopen.80432

on patterns of n bits. They verified this claim for n up to 8, while in [14] this is extended to $n \leq 10$ and in this paper up to $n \leq 28$.

Dennunzio et al. [15, 16], 2012–2013, consider ACAs, every turing machine can be simulated by an ACA, with quadratic slowdown. Introducing a certain fairness measure, they show that injectivity and surjectivity are equivalent (μ-a.a.), and the existence of a diamond is equivalent to not μ-a.a. injectivity.

Salo [17], 2014, shows that nonuniform CA generates all SFTs (subshifts of finite type) and several non-SFT sofic shifts.

2. Notations: ECA and update rules

Here, continuing the work in [10, 14], we again employ CAs as computing devices, whose work comes to an end, when the pattern transformation or function evaluation has been obtained. Also, the clocking, the temporal update rule, is completely deterministic and replaces the usual ways of representing an algorithm, either in software (initial data) or in hardware (choice of ECA and connecting graph). Thus, the algorithm resides exclusively in the clocking scheme. We therefore call functions computable in this way as "clocking-computable functions." The main additional contribution of this paper is the introduction of *unfair clocking schemes*.

2.1. Cellular automata: Neighborhoods and local update rules

We consider cellular automata (CA) on a torus or ring of n cells, that is index set $\mathbb{Z}/n\mathbb{Z}$, over the binary alphabet $\{0,1\}$. Cell index wraparound, that is, $c_i = c_j$ for $i \equiv jmodn$, and the canonical cell names are $c_{n-1}, c_{n-2}, \ldots, c_1, c_0$. We deal with elementary CAs (ECA) with three input cells, where the middle one is also the output cell.

The neighborhood (c_{i+1}, c_i, c_{i-1}) can have eight different values from $\in \mathbb{F}_2^3$. Let $k \coloneqq 4 \cdot c_{i+1} + 2 \cdot c_i + c_{i-1} \in \{0, \ldots, 7\}$. Then c_i is replaced by $c_i^+ \coloneqq p_k \in \{0,1\}$, where $p_0, \ldots, p_7$ are defined via Wolfram's rule [1] $\sum_{k=0}^{7} 2^k p_k \in \{0, \ldots, 255\}$.

The $2^{2^3} = 256$ ECAs can be arranged into 88 equivalence classes under the symbolic symmetry 0/1 and the chiral symmetry left/right ($c_{i+1} \leftrightarrow c_{i-1}$); see ([14], Appendix A). It is sufficient to consider one member per class.

We considered quad CAs (QCAs) with four inputs and nonstandard neighborhoods in ([10], Section 1.2).

Local bijectivity requires that ECA $(a,0,c) = 1-$ ECA $(a,1,c)$. This is equivalent to requiring that the hexadecimal digits of the rule be from `3, 6, 9, C`.

Example: The behavior of the ECA with Wolfram rule $57 = \mathbf{00111001}_2 = 39_{16}$:

$111 \mapsto \mathbf{0}$, $110 \mapsto \mathbf{0}$, $101 \mapsto \mathbf{1}$, $100 \mapsto \mathbf{1}$, $011 \mapsto \mathbf{1}$, $010 \mapsto \mathbf{0}$, $001 \mapsto \mathbf{0}$, $000 \mapsto \mathbf{1}$, in other words $0c_i1 \mapsto c_i^+ = c_i$, for all other contexts we have $0c_i0, 1c_i0, 1c_i1 \mapsto c_i^+ = \overline{c}_i$.

2.2. ECA: Global update rules on the torus

2.2.1. Fair update schemes

We repeat the *definition* of asynchronicity rules from ([10], Section 2).

The set AS_n of asynchronicity rules over $\mathbb{Z}/n\mathbb{Z}$ consists in all words of length n over the alphabet $\{<, \equiv, >\}$ such that both "$<$" and "$>$" occur at least once. We also include the word "$\equiv \cdots \equiv$," the synchronous case, and have.

$AS_n = (\{<, \equiv, >\}^n \setminus (\{<, \equiv\}^n \cup \{\equiv, >\}^n)) \cup \{\equiv^n\}$ with $|AS_n| = 3^n - 2^{n+1} + 2$.

A rule $\mathrm{AS} = \mathrm{AS}_{n-1} \cdots \ \mathrm{AS}_0 \in AS_n$ defines the firing order as follows:

AS_i	**Meaning**
$<$	Cell c_i fires after cell c_{i-1}
$>$	Cell c_i fires before cell c_{i-1}
$\equiv$	Cell c_i fires simultaneously with cell c_{i-1}

To ensure bijectivity, we must first have a locally bijective CA, and, furthermore, no two adjacent cells may fire simultaneously. Why this is so will be dealt with in Chapter 5. There are exactly $2^n - 2$ bijective fair rules, those from $\{<, >\}^n \setminus \{<^n, >^n\}$; see also ([10], 4.1]).

A fair update step (bijective or not) can be decomposed into a sequence of elementary steps such that all cells fire exactly once during the execution of that sequence; see [3].

2.2.2. Unfair update schemes

We now include unfair updates, where some cells may fire less often than others (even not at all).

We start with elementary steps (μ steps in [10]). During one elementary step, any nonempty subset $I \subseteq \{n-1, n-2, \ldots, 1, 0\}$ of indices may define the active cells.

These cells fire simultaneously, hence $c_i^+ = \begin{cases} ECA(c_{i+1}, c_i, c_{i-1}), & i \in I, \\ c_i, & i \notin I. \end{cases}$

We define elementary steps as words $s = s_{n-1} \cdots s_1 s_0 \in \{0,1\}^n$, with the meaning $s_i = 1$, if cell c_i fires, and $s_i = 0$ if cell c_i is inactive in this step.

A sequence of such elementary steps is upper indexed by the time step (t).

A fair update rule consists in a number of elementary steps such that every cell fires exactly once. The fair rule $as =$ "$<><><>$" for $n = 6$ can be decomposed into ($s^{(1)} = 010101,\ s^{(2)} = 101010$), while the sequence ($s^{(1)} = 010001, s^{(2)} = 101010$) is unfair, since cell c_2 does not fire at all.

The number of bijective elementary steps is the number of words of length n over the alphabet $\{0,1\}$, such that no adjacent 1's occur to ensure bijectivity. For $n = 3, 4, 5, 6$ there are 3, 6, 10, 17 such steps, respectively. The sequence obeys the law $n_k = n_{k-1} + n_{k-2} + 1$. At least for $n = 3, \ldots, 7$, the sets are as follows: prepend a 0 to each pattern of length $k-1$, prepend a 10 (a 01) to each pattern of size $k-1$ terminating in 0 (in 1), and add the new pattern 10^{k-1}. The

http://dx.doi.org/10.5772/intechopen.80432

sequence is used again in Section 5 and has been verified to coincide with OEIS A001610 up to $n = 24$.

We also can define unfair bijective rules (full steps), where we first fix some subset of size $1 \le k \le n-1$ of active cells by a word a from $\{0,1\}^n \setminus \{0^n, 1^n\}$, $a_i = 1$ meaning that cell c_i is active.

We next order adjacent active cells by the usual $<$, $>$ signs. Hence, a run of r consecutive 1's (with wraparound) has 2^{r-1} ways to fix the internal firing order. This order is independent of the other 1-runs, since the cells are separated by at least an inactive cell with $a_i = 0$. The number of bijective unfair rules on a torus of size n is

$$\sum_{p=1}^{2^n-1} \prod_{i=1}^{\#runs} 2^{r_i-1}$$

where the pattern p avoids 0^n and 1^n and then the 1-runs in the pattern have lengths $r_1, r_2, \ldots$, considering wraparound.

For $n = 3, \ldots, 7$, we have $9 = 3+6$, $30 = 4+10+16$, $90 = 5+15+30+40$, $257 = 6+21+50+84+96$, and $714 = 7+28+77+154+224+224$, such unfair rules, respectively. The terms count how many patterns with $k = 1, 2, \ldots, n-1$ active cells are feasible. These terms can be found in OEIS [18] as subsequences of A209697.

3. Patterns

We consider pattern conversions $\mathbb{F}_2^n \ni v \mapsto w \in \mathbb{F}_2^n$, where $\mathbb{F}_2^n$ can be identified with the set $\{0, 1, \ldots, 2^n - 1\}$. From Definition ([10], Def. 3), by $ECA_{\mathrm{AS}}(v) = w$, we mean that the elementary CA with rule ECA maps $v \in \{0,1\}^n$ to $w \in \{0,1\}^n$ via the asynchronicity scheme AS. We define $ECA_{\mathrm{AS}}^{\tau}(v) = ECA_{\mathrm{AS}}\big(\ ECA_{\mathrm{AS}}^{\tau-1}\ (v)\big)$ recursively for $\tau \in \mathbb{N}$, starting with $ECA_{\mathrm{AS}}^{1}\ \ (v) = ECA_{\mathrm{AS}}\ \ (v)$.

In [10, 14], we considered five universality properties (o) to (iv), where each $v \mapsto w$ makes use of a certain update rule AS applied several times. We only give a summary here. Property (iv) is ruled out for any $n \in \mathbb{N}$, while properties (o) to (iv) have only been verified experimentally, for $n \le 15$.

$$(o) \quad \exists v \in \mathbb{F}_2^n, \quad \forall w \in \mathbb{F}_2^n, \quad \exists \tau \in \mathbb{N}, \quad \exists AS \in AS_n : \ \ ECA_{\mathrm{AS}}^{\tau}(v) = w.$$

Some v is mapped to every w by varying the rule AS and the required number of time steps. There are 44 ECAs doing this.

$$(i) \quad \forall v \in \mathbb{F}_2^n, \quad \forall w \in \mathbb{F}_2^n, \quad \exists \tau \in \mathbb{N}, \ \exists AS \in AS_n : \ \ ECA_{\mathrm{AS}}^{\tau}(v) = w.$$

All v are mapped to all w by varying the rule AS and the required number of time steps. There are 6 ECAs (rules 19, 23 (for $n \not\equiv 0 \bmod 2$), 37 (for $n \not\equiv 0 \bmod 3$), 41, 57, 105 (for $n \not\equiv 0 \bmod 4$)) doing this (checked for $n \le 15$).

$$(ii) \quad \forall v \in \mathbb{F}_2^n, \quad \exists \tau \in \mathbb{F}, \quad \forall w \in \mathbb{N}_2^n, \quad \exists AS \quad \in AS_n : \ \ ECA_{\mathrm{AS}}^{\tau}(v) = w.$$

All v are mapped to all w at the same time, which time may vary for v but not for w, for different rules AS .

ECA	$n = 5$	6	7	8	9	10	11	12	13	14	15
57	445	—	70	242	35	13	13	13	13	13	10
105	—	—	570	—	14	—	6	—	6	—	8

Table 1. Minimum time τ required to satisfy (iii).

ECA-57 realizes this for $n = 5, \ldots, 15$ (no further n has been considered).

ECA-105 realizes this for odd $n = 7, 9, 11, 13, 15$ (no further n has been considered).

$$(iii) \quad \exists \tau \in \mathbb{N}, \quad \forall v \in \mathbb{F}_2^n, \quad \forall w \in \mathbb{F}_2^n, \quad \exists AS \quad \in AS_n : \quad ECA_{\mathrm{AS}}^{\tau}(v) = w.$$

All v are mapped to all w at the same time, varying the rule AS. This is actually possible for the two survivors of property (ii); see **Table 1**. The required time roughly decreases with growing n, since we have $3^n - 2^{n+1} + 1$ rules to choose from for 2^n patterns w. Thus for higher n the probability to meet the conversion early on increases.

$$(iv) \quad \exists \tau_0 \in \mathbb{N}, \quad \forall \tau \geq \tau_0, \quad \forall v, w \in \mathbb{F}_2^n, \quad \exists AS \quad \in AS_n : \quad ECA_{\mathrm{AS}}^{\tau}(v) = w.$$

Eventually, all conversions may happen at all times for some update scheme. This property cannot be satisfied, for no ECA; see ([14], Thm. 2).

For more details and results for QCAs consult Section 2 of [14] and Section 3 of [10].

4. Bijective functions

We first introduce the computer algebra system GAP and then give several examples.

4.1. GAP: Graphs, algorithms, programming

4.1.1. GAP and the alternating group A_{2^n}

GAP [19] is a system for computational discrete algebra, in particular computational group theory. We use GAP to decide, whether certain fair or unfair update rules generate the full symmetric or alternating group S_{2^n} or A_{2^n}, respectively.

Our results so far:

Theorem 1.

i. The fair update rules for ECA-57 generate the full symmetric group S_8 for $n = 3$.

ii. The fair update rules for ECA-57 generate the full alternating group A_{2^n} for $n = 4, \ldots, 11$.

http://dx.doi.org/10.5772/intechopen.80432

iii. The (unfair) elementary update rules with exactly one active cell for ECA-57 generate the full alternating group A_{2^n} for $n = 4, \ldots, 28$.

Proof.

(i) By exhaustive generation of all $8! = 40320$ permutations on $\{000, 001, \ldots, 111\}$.

(ii, iii) First observe that all these rules consist of an even number of transpositions. Hence, we will at most obtain the alternating group A_{2^n}. This group comprises those bijective functions on $\{0, 1, \ldots, 2^n - 1\}$ which have positive sign as a permutation.

We checked (ii) with GAP for certain sets of 5 fair rules for each of these sizes n, and GAP's function `IsNaturalAlternatingGroup(G) returned true`.

For (iii) we have the canonical set of n elementary unfair update rules, with exactly one cell active in each rule. This set generates all fair and unfair update rules and hence is sufficient to decide on the group generated by all rules.

Again, GAP shows that indeed the alternating group is generated, for $n = 4, \ldots, 28$. We used 144 GB of RAM, which was sufficient for $n = 28$ but not so for $n = 29$. □

We believe that, apart from the special case $n = 3$, we always obtain the alternating group.

Conjecture.

For every torus size $n \in \mathbb{N}, n \geq 4$, both the $2^n - 2$ fair update rules, as well as the n elementary unfair update rules with a single active cell, are a generating set for the full alternating group on 2^n elements, using ECA-57.

4.1.2. Example for GAP usage with $n = 4$

We replaced 0 by 2^n, since GAP only uses numbers from $\mathbb{N}$. P00 to P03 are the permutations generated by the elementary steps $s = 0001, 0010, 0100$, and 1000, respectively.

```
mjv@Panda ~/GAP $gap -b
gap> P00 := (16,1)(2,3)(4,5)(6,7)(10,11)(14,15);
(1,16)(2,3)(4,5)(6,7)(10,11)(14,15)
gap> P01 := (16,2)(4,6)(5,7)(8,10)(12,14)(13,15);
(2,16)(4,6)(5,7)(8,10)(12,14)(13,15)
gap> P02 := (16,4)(1,5)(8,12)(9,13)(10,14)(11,15);
(1,5)(4,16)(8,12)(9,13)(10,14)(11,15)
gap> P03 := (16,8)(1,9)(2,10)(3,11)(5,13)(7,15);
(1,9)(2,10)(3,11)(5,13)(7,15)(8,16)
gap> G04 := Group(P00,P01,P02,P03);;
gap> IsNaturalAlternatingGroup(G04);
true
gap> Size(G04);
10461394944000
```

4.1.3. Example for GAP usage with $n = 5$

```
mjv@Panda ~/GAP $cat G5
P00 := (32,1)(2,3)(4,5)(6,7)(8,9)(10,11)(12,13)(14,15)
(18,19)(22,23)(26,27)(30,31);
P01 := (32,2)(4,6)(5,7)(8,10)(12,14)(13,15)(16,18)
(20,22)(21,23)(24,26)(28,30)(29,31);
P02 := (32,4)(1,5)(8,12)(9,13)(10,14)(11,15)(16,20)
(17,21)(24,28)(25,29)(26,30)(27,31);
P03 := (32,8)(1,9)(2,10)(3,11)(16,24)
(17,25)(18,26)(19,27)(20,28)(21,29)(22,30)(23,31);
P04 := (32,16)(1,17)(2,18)(3,19)(4,20)(5,21)(6,22)
(7,23)(9,25)(11,27)(13,29)(15,31);
G05 := Group(P00,P01,P02,P03,P04);
```

```
mjv@Panda ~/GAP $gap -b
gap> Read("G5");
gap> IsNaturalAlternatingGroup(G05);
true
gap> Size(G05);
131565418466846765083609006080000000
gap> quit;
```

Here, $1.315\ldots \cdot 10^{35} = 32!/2$.

4.1.4. Example for GAP usage with $n = 28$

We define the permutations P00 to P27 via a C++ − program; see Appendix A. Using it like gap.out 28 > G28, its output, the file G28, is then read in by GAP.

```
mjv@turing:/var/GAP$ gap -b -m 140G
gap> Runtime();Read("G28");Runtime();IsAlternatingGroup(G28);Runtime();
IsNaturalAlternatingGroup(G28);Runtime();
5592
1639756
true
8915688
true
8915688
gap>
```

Times are in milliseconds. Therefore, reading in the 56 GB of permutations P00 to P27 generating the group G28 took 1640 seconds, or about half an hour, while actually checking the resulting group for $A_{2^{28}}$ took another 7280 seconds or 2 hours. The same procedure for G29 resulted in lack of memory; 140 GB of RAM are not sufficient.

http://dx.doi.org/10.5772/intechopen.80432

Recall that, according to Stirling's formula, $A_{2^{28}}$ has $2^{28}!/2 \approx \left(\frac{2^{28}}{e}\right)^{2^{28}} \approx 10^{10^9}$ elements, a number with about 1 billion (1000^3) digits. Amazingly, GAP gets it done!

4.2. Examples for bijective functions

4.2.1. Multiplication by 9 mod 16

We give two realizations of the function *byNine* : $x \mapsto 9x$ mod 16 in **Table 2**.

Observe that the first 23 steps only implement the permutation $(08)(2A)$, consisting of two transpositions. The final 24th step is almost identical to the whole function.

First realization in hexadecimal	1. Active cells	2. Active cells	Second realization in hexadecimal	
0xFEDCBA9876543210	0101	0100	0xFEDCBA9876543210	(Id)
0xAB98EFDC67012345	1010	1000	0xBA98FEDC76103254	
0x0312C57E4DA98B6F	0101	0001	0x32107E5CF698BAD4	
0x5243806B19FDCE7A	1000	1000	0x23016F4CE798ABD5	
0xDA4B08639175CEF2	0001	0001	0xAB89674CEF10235D	
0xDB5A18729064CFE3	0010	1010	0xBA89765CFE01324D	
0xFB781A509246EDC3	0100	0101	0x3021D4FE5CA9B867	
0xBF7C5E14D206A983	1000	0010	0x253491AB08FDEC76	
0x37FCDE945A86210B	0001	0001	0x0736918B2ADFCE54	
0x26ECDF954B87301A	1010	1010	0x1627908A3BDECF45	
0x84CE751F632DBA90	0101	0101	0x948D1A20B37CE56F	
0xC18B604A7239EFD5	1010	1010	0xD1C94F35E268B07A	
0xE9234A60D8B1C57F	0101	0101	0x79E165BFC8423AD0	
0xBD321F759CE4806A	1010	1010	0x6DB470EA8C132F95	
0x37B895DF1EC62A40	0100	0100	0x4736DAC02E9B851F	
0x37FCD19B5A862E04	1000	0010	0x07369E842ADFC15B	
0xBF7C5913D206AE84	0100	0101	0x25349CA608FDE17B	
0xFB781D539246EAC0	0010	1010	0x3021D8F75CA9B46E	
0xDB5A1F739064C8E2	0001	0001	0xBA89725DFE01364C	
0xDA4B0E629175C8F3	1000	1000	0xAB89634DEF10275C	
0x52438E6A19FDC07B	0101	0001	0x23016B45E798AFDC	
0x0312CB7F4DA9856E	1010	1000	0x32107A54F698BEDC	
0xAB98E3D567012F4C	0101	0100	0xBA98F2D476103E5C	
0xFEDCB29076543A18	1000	1000	0xFEDCB29076543A18	(08)(2A)
0x7E5C3A18F6D4B290			0x7E5C3A18F6D4B290	

Table 2. Multiplication $x \mapsto 9x$ in $\mathbb{Z}/16\mathbb{Z}$.

Active cells	FED...210 → 62C...931 some patterns in binary							All values F...0 in hexadecimal
0001	1111	1110	1101	…	0010	0001	0000	0xFEDCBA9876543210
1010	1110	1111	1101	…	0011	0000	0001	0xEFDCAB9867452301
0101	1100	0101	0111	…	1011	1010	1001	0xC57E03124D6F8BA9
1010	1000	0000	0110	…	1110	1111	1101	0x806B5243197ACEFD
0101	0010	1010	0100	…	1100	0101	0111	0x2A43F86B91D0EC57
1000	0011	1111	0001	…	1000	0000	0110	0x3F12AC7ED495B806
0100	1011	0111	1001	…	0000	1000	0110	0xB79A2CFE541D3086
1000	1111	0111	1101	…	0100	1100	0110	0xF7DE28BA105934C6
0001	0111	1111	0101	…	0100	1100	0110	0x7F5EA03298D1B4C6
1000	0110	1110	0100	…	0101	1100	0111	0x6E4FB12398D0A5C7
0100	0110	1110	0100	…	1101	1100	1111	0x6E4739AB10582DCF
1010	0110	1010	0000	…	1001	1000	1011	0x6A073DEF541C298B
0100	0100	0000	1010	…	0001	0010	0011	0x40ADB7C5F69E8123
1010	0000	0100	1110	…	0101	0010	0011	0x04E9F781B6DAC523
0001	1010	0110	1100	…	1111	1000	1011	0xA6C15D293470EF8B
1000	1011	0111	1100	…	1110	1000	1010	0xB7C04D392561FE8A
0101	0011	1111	1100	…	1110	0000	0010	0x3FC845B1AD697E02
1000	0010	1010	1000	…	1011	0101	0011	0x2A8C10E4F97D6B53
0100	1010	0010	0000	…	0011	1101	1011	0xA20C98E471F563DB
1000	1110	0010	0100	…	0011	1001	1111	0xE248DCA075B1639F
0100	1110	1010	0100	…	1011	0001	0111	0xEA405C28FD396B17
1010	1010	1110	0000	…	1111	0101	0111	0xAE04182CB93D6F57
0100	0000	1100	1010	…	0101	1111	1101	0x0CA6928E31B745FD
1010	0100	1000	1110	…	0001	1011	1001	0x48E6D2CA35F701B9
	0110	0010	1100	…	1001	0011	0001	0x62C478E0BF5DA931

Table 3. Exponentiation $x \mapsto 3^x$ in $\mathbb{F}_{17}$.

Hence *byNine* = `(19)(3B)(5D)(7F)` = `[(08)(2A)][(08)(19)(2A)(3B)(5D)(7F)]`, where the first bracket requires 23 steps, and the second is the elementary rule 1000 (only the leftmost cell is active). The difference between the two realizations (sequences $s^{(1,\ldots,24)}$ in hex), where the outer parentheses are inverses of each other and the inner part is self-inverse, and their difference:

```
1.: (5A581248)1(A5A5A)4(842185A5)8
Diff 124-81A——-A18-421-
2.: (48181A52)1(A5A5A)4(25A18184)8
```

http://dx.doi.org/10.5772/intechopen.80432

4.2.2. Exponentiation and logarithm in $\mathbb{F}_{17}$

Identifying 16 with 0b0000, we can map $\mathbb{F}_{17}^*$ to $\{0,1\}^4$. Here is the exponentiation $x \mapsto 3^x$; see **Table 3**.

`FEDCBA9876543210` $\rightarrow \ldots$
$s^{(1,\ldots,24)}$ = [0001, 1010, 0101, 1010, 0101, 1000, 0100, 1000, 0001, 1000, 0100, 1010, 0100, 1010, 0001, 1000, 0101, 1000, 0100, 1000, 0100, 1010, 0100, 1010]
$\ldots \rightarrow$`62C478E0BF5DA931`

The inverse function $x \mapsto \log_3(x)$ is therefore computed by the inverse update sequence $s^{(1,\ldots,24)}$ = [1010, 0100, 1010, 0100, 1000, 0100, 1000, 0101, 1000, 0001, 1010, 0100, 1010, 0100, 1000, 0001, 1000, 0100, 1000, 0101, 1010, 0101, 1010, 0001].

5. Non-bijective functions

5.1. Non-bijective global rules and in-degree distributions

In Section 2.2, we have said that, in order to ensure bijectivity, we must first have a locally bijective CA, and furthermore, no two adjacent cells may fire simultaneously. Here is why:

Example.

We start with the effect of AS_i = "$\equiv$" for the locally bijective ECA-57. We consider four adjacent cells and the effect of $\equiv$ between the two middle cells, which are thus updated simultaneously, $s = 0110$.

$v \mapsto$ **ECA-57**$_{\mathbf{0\equiv 00}}(v)$											
0000	↦	0110	0001	↦	**0101**	1000	↦	1110	1001	↦	1101
0010	↦	0000	0011	↦	**0011**	1010	↦	1100	1011	↦	1111
0100	↦	0010	0101	↦	**0011**	1100	↦	1010	1101	↦	1011
0110	↦	0100	0111	↦	**0101**	1110	↦	1000	1111	↦	1001

We obtain the patterns 0011 and 0101 twice, while missing 0001 and 0111. Hence the image is smaller than the full 2^4 by 2 or by a factor 7/8 in general.

We have the following in-degree distribution (loss pattern 4 of [[10], Table 8]):

#(domain)	#(range)	In-degree	Distribution φ
12	12	1	$\varphi(1) = 12$
4	2	2	$\varphi(2) = 2$
0	2	0	$\varphi(0) = 2$

Twelve patterns map bijectively (1,1) to 12 patterns, 4 patterns map 2:1 to 2 patterns, while 2 patterns of the range are not met at all (0,1).

We have #(domain) = in-degree×#(range) for every in-degree. Also, the overall sum is

$$\sum \#(\text{domain}) = \sum \text{ in-degree} \times \#(\text{range}) = 2^n.$$

Any additional simultaneous firing may increase the losses. Additional in-degree distributions (loss patterns) for ECAs and QCAs may be found in ([10], Table 8).

We restate Theorem 3 (i) from [14]:

Theorem 2.

We assume an ECA that generates at least the alternating group A_{2^n}, when using temporal rules from $\{<,>\}^n \setminus \{<^n,>^n\}$. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be any non-bijective function on n symbols, where we require $n \geq 4$ for ECA.

Let $\#(w) = |\{v|f(v) = w\}|$ be the number of configurations v leading to configuration w. Then f is clocking-computable by an ECA with in-degree distribution $\varphi(0,1,2) = (12,2,2)$ or $(24,4,4)$, if and only if

$$\sum_{w \in \mathbb{F}_2^n} \lfloor \#(w)/2 \rfloor \geq \varphi(2) \cdot 2^n / \sum_{k=0}^{2} \varphi(k).$$

Proof. See [14], Theorem 3. □

5.2. Algorithm for non-bijective functions

The computation of a non-bijective function can be decomposed into 3 steps:

Step I. We start in the middle: Shrink the 2^n singletons of the domain to the desired distribution on the range. Find a sequence of elementary steps, necessarily including non-bijective ones, which generates the same distribution of counts in the image space as for the original function. Usually there are more than one of these sequences with the same complexity. Values/patterns as such do not yet play a role; therefore the requirements are usually easy to meet in a variety of ways, and the sequence of necessary elementary steps is short.

Step II. Bijectively map the input of the original function to the input of any of the results of Step I in such a way that the desired image counts are matched. Only the occurence counts must be observed, while, within this restriction, pattern values can be permuted.

Step III. Bijectively map the output from Step I to the output of the desired function, considering the flow within Steps II and I in that order. Now all values matter and cannot be permuted. However, this step takes place on a subset of size $|Im(f)|$ instead of 2^n.

In Steps II and III we use the meet-in-the-middle approach, applying update rules starting both from the input (identity) and the desired function values to take advantage of the birthday paradox: In this way, $2 \cdot \sqrt{\#}$ patterns are sufficient to generate $\sqrt{\#} \cdot \sqrt{\#} = \#$ potential matches in the middle (here $\# \approx |A_{2^n}|$), a considerable saving in both space and time.

http://dx.doi.org/10.5772/intechopen.80432

5.3. Example of a non-bijective function multiplication up to $3 \times 3 = 9$ on the torus of size $n = 4$

The 4-bit input is interpreted as a pair of numbers from the set $\{0, 1, 2, 3\}$, whose product, the output, lies within the set $\{0, 1, 2, 3, 4, 6, 9\}$.

In	Out	In	Out	In	Out	In	Out
00.00	0000	01.00	0000	10.00	0000	11.00	0000
00.01	0000	01.01	0001	10.01	0010	11.01	0011
00.10	0000	01.10	0010	10.10	0100	11.10	0110
00.11	0000	01.11	0011	10.11	0110	11.11	1001

5.3.1. Step I

The image consists of 7 patterns with occurence counts 7 (0000), 2 (0010,0011,0110), and 1 (0001,0100,1001).

In Step I, we thus have to shrink the set of patterns present from 16 to 7 in such a way that (any) 7 patterns are mapped to the same one, and additionally 3 sets of two patterns are joined within each set. Finally, the three remaining input patterns stay on their own.

An exhaustive search over sequences with 4 update rules, starting and ending with a non-bijective one, yields 72 such sequences with the desired shrinking factor, one example is the sequence $s^{(1,\ldots,4)} = 0011, 1101, 0110, 0111$ of active cells.

All patterns in hex	Active cells	Occurrence counts				
		1	2	3	5	7
0123.4567.89AB.CDEF	0011	16	0	0	0	0
3012.7654.A99A.EFDC	1101	14	2	0	0	0
ADCB.E781.7557.B218	0110	5	4	1	0	0
CBAF.85E5.5335.F05E	0111	5	3	0	1	0
AED8.E292.2222.8729		3	3	0	0	1

5.3.2. Step II

After Step I, the output pattern 2 has frequency 7. Therefore, 2 has to match 0 in Step III, while 0,1,2,3,4,8,C are matched to 5,7,8,9,A,B,E in any order in Step II. Thus, we already have 7! different possibilities for Step II.

Similarly, we can maps 0110 and 1001 to either $\{1, 4\}$, $\{3, C\}$, or $\{6, F\}$ and so forth. Let c run through the occurrence counts, here $c = 7$, 2 and 1 are actually taken, and let then $p(c)$ be the number of output patterns with in-degree c, here $p(7) = 1$, $p(2) = p(1) = 3$. We get

$$\prod_c c!^{p(c)} p(c)! = 7!^1 1! \cdot (2!)^3 3! \cdot (1!)^3 3! = 1451520$$

bijective functions consistent with the count distribution to choose from in Step II. One of these is given in the left column of **Table 4**.

5.3.3. Step III

The outcome of Steps I + II completely fixes the necessary permutation for Step III. However, we do not have to deal with 2^n values, but only $|\mathrm{Im}(f)|$ are relevant, in our example 7 instead of

Step II		**Step I**		**Step III**	
0xFEDCBA9876543210	0001	0x0123456789ABCDEF	0011	0x2AE879D	0001
0xEFDCAB9867452301	1010	0x30127654A99AEFDC	1101	0x3BF869D	0010
0xC57E03124D6F8BA9	0100	0xADCBE7817557B218	0110	0x3BDA49F	0100
0x817A4352096BCFED	0010	0xCBAF85E55335F05E	0111	0x3F9E0DB	1010
0xA1586370294BEDCF	0001	0xAED8E29222228729		0xB51CA73	0101
0xB0487261395AFDCE	1010			0xE048F62	0010
0x3A62D849B1F057EC	0100			0xC26AD40	0101
0x3E629C0DF5B417A8	0010			0x837F915	0010
0x3C409E2FD7B6158A	0100			0xA35D917	0101
0x3804DA2B97F651CE	1010			0xF209D46	1000
0xB2A670831D54F9EC	0001			0x7A81546	0100
0xA3B761820D45E9FC	1000			0x7EC5106	1010
0x2B3F690A854DE17C	0100			0xDCEF9A4	0101
0x2F3B6D4EC109A578				0x98BADF1	1000
				0x1032579	0001
				0x0123469	

Table 4. Steps II, I, and III for multiplication up to $3 \times 3 = 9$.

IN	**Step II**	**Step I**	**Step III**	**IN**	**Step II**	**Step I**	**Step III**
0010	5	2	0	0110	1	E	2
0001	7	2	0	1001	4	E	2
0000	8	2	0	1101	3	8	3
0100	9	2	0	0111	C	8	3
0011	A	2	0	1010	D	7	4
1100	B	2	0	1011	6	9	6
1000	E	2	0	1110	F	9	6
0101	0	A	1	1111	2	D	9

Table 5. Steps II, I, and III by output patterns.

http://dx.doi.org/10.5772/intechopen.80432

16. Again, meet-in-the-middle yields a match. Be aware that the necessary effort does not drop from 16! to 7! but only to $\frac{16!}{(16-7)!} \approx 11!$ or $\frac{2^n!}{(2^n-|Im(f)|)!}$ in general.

In total, 13 + 4 + 15 = 32 steps are required to compute this multiplication function (**Table 5**).

6. Efficiency

In the case of unfair bijective functions, any subset of cells may fire simultaneously, provided that no adjacent cells are contained in the set.

We define *local efficiency* of a bijective update sequence by two properties:

- No cell is active during two consecutive time steps (these two would cancel each other).
- No active cell can be moved to the previous time step.

Global efficiency—which shortest update sequence generates a certain function/permutation—is beyond the scope of this paper (it is dealt with implicitly by brute force in a breadth-first manner).

Lemma 3.

A sequence of rules is bijective and efficient, if $s_k^{(t)} = 1$ (cell c_k active in step t) implies that.

i. $s_{k-1}^{(t)} = 0$ and $s_{k+1}^{(t)} = 0$ (both inactive),

ii. $s_k^{(t-1)} = 0$ (inactive), and.

iii. at least one of $s_{k-1}^{(t-1)}$ and $s_{k+1}^{(t-1)}$ equals 1, active.

Proof.

(i) is required for and then ensures bijectivity.

If $s_k^{(t-1)}$ was active, both $s_k^{(t-1)}$ and $s_k^{(t)}$ could be removed, since each single-location action is an involution. Thus (ii) is required.

If $s_{k-1}^{(t-1)}$ and $s_{k+1}^{(t-1)}$ were both inactive, the active cell $s_k^{(t)}$ could be moved to $s_k^{(t-1)}$, maintaining bijectivity. Hence (iii) is required. □

Using this lemma, we now can compute an upper bound on the number of bijective functions realizable with t update steps: We define a matrix A with rows and columns indexed by the bijective elementary update rules, here named $s(1), s(2), \ldots$. Set $a_{ij} := 1$, if rule $s(i)$ followed by rule $s(j)$ is efficient, $a_{ij} := 0$ otherwise.

Let $\lambda(n)$ be the largest eigenvalue of A.

Example

	A_{ij} **for** $n = 5$**:**									
$s(i) \setminus s(j)$	$s(1)$	$s(2)$	$s(3)$	$s(4)$	$s(5)$	$s(6)$	$s(7)$	$s(8)$	$s(9)$	$s(10)$
$s(1)$ = 00001	0	1	0	0	0	0	0	1	1	0
$s(2)$ = 00010	1	0	1	1	0	0	0	0	0	0
$s(3)$ = 00100	0	1	0	0	1	0	1	0	0	0
$s(4)$ = 00101	0	1	0	0	1	0	1	1	1	0
$s(5)$ = 01000	0	0	1	0	0	0	0	1	0	1
$s(6)$ = 01001	0	1	1	0	0	0	0	1	1	1
$s(7)$ = 01010	1	0	1	1	0	0	0	1	0	1
$s(8)$ = 10,000	1	0	0	0	1	1	0	0	0	0
$s(9)$ = 10,010	1	0	1	1	1	1	0	0	0	0
$s(10)$ = 10,100	1	1	0	0	1	1	1	0	0	0

Observe that 10,100 followed by 00001 is efficient, while the other way round 00001 leaves room to move $s_2^{(2)}$ up to $s_2^{(1)}$. Hence, A is not symmetrical for $n \geq 5$.

In **Table 6**, we denote several figures describing the number of efficient bijective update schemes (including unfair ones). The number of rules is denoted by $\#S(n)$; it can be found as A001610 in OEIS [18]; see also Section 2.2.2. The number of nonzero entries in the matrix A is $\#A(n)$. The quotient $\#A(n)/\#S(n)$ is the average number of 1's in each row/column.

We denote by $\#_n(t)$ the number of effective bijective update schemes of length t on a torus of size n. Essentially, $\#_n(t) = \Theta\left(\lambda(n)^t\right)$.

Let $\rho(n) = \lim_{t\to\infty} \frac{\#_n(t)}{\lambda(n)^t}$. Then $\rho(n)$ gives the constant hidden in the Landau symbol Θ.

We have $\#_n(t)/\left(\lambda(n)^t \cdot \rho(n)\right) \to 1$.

Finally,

$$T(n) = \log_{\lambda(n)}(2^n!/(2 \cdot \rho(n)))$$

is the number of steps necessary such that $\#_n(T(n)) \approx |A_{2^n}| = 2^n!/2$. Invoking the birthday paradox by the meet-in-the-middle approach, we therefore expect to require $T(n)/2$ steps each, starting from the initial identity permutation and backwards from the desired function, to achieve a match yielding the function evaluation.

The asymptotic behavior, as far as we can infer from the range $n = 3, \ldots, 24$, is

$$\#S(n) = \Theta(\varphi^n), \varphi = 1.618\ldots$$

$$\lambda(n) = \Theta(1.3^n)$$

http://dx.doi.org/10.5772/intechopen.80432

n	$\#S(n)$	$\#A(n)$	$\left\lfloor \frac{\#A(n)}{\#S(n)} \right\rfloor$	$\lambda(n)$	$\rho(n)$	$\lfloor \log(\lvert A_{2^n} \rvert) \rfloor$	$\lfloor T(n) \rfloor$
3	3	6	2	2.000	1.500	10	14
4	6	18	3	3.000	2.000	30	27
5	10	40	4	3.732	2.887	81	61
6	17	98	5	5.000	4.000	204	126
7	28	224	8	6.236	6.261	495	270
8	46	514	11	8.522	7.044	1166	544
9	75	1158	15	10.697	11.581	2685	1132
10	122	2602	21	14.599	11.142	6077	2266
11	198	5808	29	18.288	20.464	13,571	4668
12	321	12,930	40	24.941	17.141	29,978	9319
13	520	28,704	55	31.253	35.107	65,630	19,065
14	842	63,618	75	42.634	25.696	142,612	38,002
15	1363	140,806	103	53.426	58.716	307,933	77,402
16	2206	311,362	141	72.878	37.982	661,287	154,189
17	3570	688,024	192	91.321	96.473	1.413e + 06	313,092
18	5777	1,519,586	263	124.571	55.699	3.008e + 06	623,547
19	9348	3,354,944	358	156.098	156.314	6.380e + 06	1.26e + 06
20	15,126	7,405,058	489	212.932	81.313	1.348e + 07	2.51e + 06
21	24,475	16,341,254	667	266.822	250.502	2.842e + 07	5.08e + 06
22	39,602	36,056,154	910	363.969	118.336	5.976e + 07	1.01e + 07
23	64,078	79,547,616	1241	456.085	397.758	1.253e + 08	2.04e + 07
24	103,681	175,485,442	1692	622.138	171.942	2.623e + 08	4.07e + 07

Table 6. Efficiency related values for $n = 3, \ldots, 24$.

$$T(n) = 2.4 \cdot 2^n \cdot (1 + o(1)).$$

We can compare the asymptotic number $T(n)$ of elementary steps necessary for unfair update rules with the lower bound on the (full) steps for fair rules.

Lemma 4 ([10], Section 5, Lemma 1(i)).

There are clocking-computable bijective functions that require at least

$$\lceil \log(2^n!) / \log(2^n - 2) \rceil = 2^n + O(1)$$

steps or evaluations per cell.

Remarkably, as far as the experimental results for $n \leq 24$ indicate, the number $T(n)$ of elementary steps is only a constant factor (≈ 2.4) higher than the lower bound for full update steps.

If that result remains valid for all $n \in \mathbb{N}$, and the reported values strongly suggest this, this would mean that the typical full step can be replaced by no more than 2..3 elementary unfair steps, independent of n.

As can be appreciated, $n = 5$ might be feasible for an attack by meet-in-the-middle, but $n = 6$ and above is certainly no candidate for this brute-force approach. Dealing with these sizes will require a more intelligent approach, for instance using group-theoretic techniques like representation theory, applied to the alternating group.

7. Conclusion

We have seen that many functions are clocking-computable, namely, the even bijective ones and the non-bijective ones with enough "loss" in their image.

The elementary cellular automaton ECA-57 can be used to implement an etherealware computing device: The computed function is a result only of the clocking order.

Temporal order of activating the CA cells is thus a new way to encode algorithms, a "volatilization of information".

We increased the size for example programs from $n = 3$ in [10, 14] to $n = 4$ and are confident to also be able to tackle the case $n = 5$.

From $n = 6$ onwards, we shall need more mathematical concepts, e.g., from group theory.

Acknowledgements

My thanks go to Dr. Mónica del Pilar Canales Chacón for proofreading, commenting, and all the rest. Furthermore, the anonymous referee gave valuable comments concerning the first draft of the paper, pointing out a substantial error and several occasions for clarifying the intended meaning.

Appendix A. gap.cc

```
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace std;

int main(int argc, char** args) {

  long long N = atoi(args[1]); // size of torus

  const long long EN = 1LL<<N; // 2^N
```

http://dx.doi.org/10.5772/intechopen.80432

```
  long long ix;

  cout << "P00 := (" << (1LL<<N) << ",1)";
  ix = 0; // rightmost cell active, n=0
  for (long long i = 1; i < EN; i++) {
    if (((i ^ 1LL) > i) (((5LL<<(N-1LL))
        & (i ^ (i<<N))) != (1LL<<(N-1LL)))) {
      cout << "(" << i << "," << (i^1LL) << ")";
      ix++;
      if ((ix%6LL) == 0) {cout << endl ;}
    }
  }
cout << ";" << endl;

for (long long n = 1; n < N; n++) { // active cell/bit
  long long En = 1LL<<n;
  ix = 0;
  cout << "P" << (char) (0x30+((n/10)%10))
       << (char) (0x30+(n%10)) << " := ("
       << (1LL<<N) << "," << (1LL<<n) << ")";

  for (long long i = 1; i < EN; i++) // all 2^N patterns
    if (((i ^ En) > i) && (((0x5LL<<(n-1LL)) & (i ^ (i<<N)))
        != (1LL<<(n-1LL)))) {
      cout << "(" << i << "," << (i^En) << ")";
      ix++;
      if ((ix%6LL) == 0) {cout << endl ;}
    }
  }
  cout << ";" << endl;
} // n

cout << "G" << (char) (0x30+((N/10)%10))
     << (char) (0x30+(N%10)) << " := Group(P00" ;

for (int n = 1; n < N; n++) {
  cout << ",P" << (char) (0x30+((n/10)%10)) << (char) (0x30+(n%10));
}
cout << ");" << endl;

//remainder, what to put interactively in GAP, no use as part of file
cout << "#IsNaturalAlternatingGroup(G"<< (char) (0x30+((N/10)
     << (char) (0x30+(N/10)%10)) << " ); " << endl ;
cout << "#IsNaturalSymmetricGroup(G"<< (char) (0x30+((N/10) %10))
     << (char) (0x30+(N%10)) << " ); " << endl ;
cout << "#Size(G"<< (char) (0x30+((N/10)%10))
     << (char) (0x30+(N%10)) << " ); " << endl ;
}
```

Author details

Michael Vielhaber[1,2]*

*Address all correspondence to: vielhaber@gmail.com

1 HS Bremerhaven, Bremerhaven, Germany

2 Universidad Austral de Chile, Instituto de Cs. Físicas y Matemáticas, Valdivia, Chile

References

[1] Wolfram S. Cellular automata as models of complexity. Nature. 1984;**311**:419-424

[2] Ingerson TE, Buvel RL. Structure in asynchronous cellular automata. Physica. 1984;**10D**: 59-68

[3] Barrett CL, Reidys CM. Elements of a theory of computer simulation I: Sequential CA over random graphs. Applied Mathematics and Computation. 1999;**98**:241-259

[4] Barrett CL, Mortveit HS, Reidys CM. Elements of a theory of simulation II: Sequential dynamical systems. Applied Mathematics and Computation. 2000;**107**:121-136

[5] Barrett CL, Mortveit HS, Reidys CM. Elements of a theory of simulation III: Equivalence of SDS. Applied Mathematics and Computation. 2001;**122**:325-340

[6] Barrett CL, Mortveit HS, Reidys CM. ETS IV: Sequential dynamical systems: Fixed points, invertibility and equivalence. Applied Mathematics and Computation. 2003;**134**:153-171

[7] Siwak P. Filtrons of automata. Proc. UMC 2002, unconventional models of computation. LNCS. 2002;**2509**:66-85

[8] Lee J, Peper F, Adachi S, Morita K, Mashiko S. Reversible computation in asynchronous cellular automata. Proceedings of Unconventional Models of Computation, LNCS. 2002; **2509**:20-229

[9] Laubenbacher P. Update schedules of sequential dynamical systems. Discrete Applied Mathematics. 2006;**154**:980-994

[10] Vielhaber M. Computing of functions on *n* bits by asynchronous clocking of cellular automata. Natural Computing. 2013;**12**:307-322. DOI: 10.1007/s11047-013-9376-7

[11] Fatès N, Thierry E, Morvan M, Schabanel N. Fully asynchronous behavior of double-quiescent elementary cellular automata. TCS. 2006;**362**:1-16

[12] Macauley M, McCammond J, Mortveit HS. Order Independence in Asynchronous Cellular Automata. 2007. arXiv:0707.2360v2 [math.DS]

[13] Macauley M, McCammond J, Mortveit HS. Dynamics groups of asynchronous cellular automata. Journal of Algebraic Combinatorics. 2010;**33**:31-55

[14] Vielhaber M. Computing by temporal order: Asynchronous cellular automata. Formenti E, editor. EPTCS 2012. Proceedings of Automata & JAC. Vol. 90; 2012. arXiv:1208.2762. DOI: 10.4204/EPTCS.90.14

[15] Dennunzio A, Formenti E, Manzoni L, Mauri G. Computing issues of asynchronous cellular automata. Fundamenta Informaticae. 2012;**120**:165-180

[16] Dennunzio A, Formenti E, Manzoni L, Mauri G. M-asynchronous cellular automata: From fairness to quasi-fairness. Natural Computing. 2013;**12**(4):561-572

[17] Salo V. Realization problems for nonuniform cellular automata. TCS. 2014;**559**. DOI: 10.1016/j.tcs.2014.07.031

[18] The On-Line Encyclopedia of Integer Sequences https://oeis.org

[19] The GAP Group, GAP—Groups, Algorithms, and Programming, Version 4.9.1; 2018. www.gap-system.org

IntechOpen