Computational Hierarchy of Elementary Cellular Automata

Barbora Hudcová[1,2] and Tomáš Mikolov[2]

[1]Charles University, Prague
[2]Czech Institute of Informatics, Robotics and Cybernetics, CTU, Prague

Abstract

The complexity of cellular automata is traditionally measured by their computational capacity. However, it is difficult to choose a challenging set of computational tasks suitable for the parallel nature of such systems. We study the ability of automata to emulate one another, and we use this notion to define such a set of naturally emerging tasks. We present the results for elementary cellular automata, although the core ideas can be extended to other computational systems. We compute a graph showing which elementary cellular automata can be emulated by which and show that certain chaotic automata are the only ones that cannot emulate any automata non-trivially. Finally, we use the emulation notion to suggest a novel definition of chaos that we believe is suitable for discrete computational systems. We believe our work can help design parallel computational systems that are Turing-complete and also computationally efficient.

Introduction

Discrete systems exhibit a wide variety of dynamical behavior ranging from ordered and easily predictable to a very complex, disordered one. In this paper, we study cellular automata (CA) that have intriguing visualizations of their space-time dynamics. Observing them helps us build intuition for distinguishing the different types of dynamics, as studied by Wolfram (2002). Informally, complex CA produce higher-order structures, whereas the space-time diagrams of chaotic CA are seemingly random. Though CA dynamics has been studied extensively (Wolfram (1984), Kůrka (2009), Wuensche and Lesser (2001), Gutowitz (1990), Zenil (2009)), it is still a very difficult problem to formally define the notions of complexity and chaos in CA.

Traditionally, complexity of CA is studied through their computational capacity (Toffoli (1977), Cook (2004)), whereas chaos is studied via topological dynamics (Devaney (1989)). As the two approaches are not connected in any obvious way, it is an interesting open problem of whether chaotic CA can compute non-trivial tasks (Martinez et al. (2013)).

In this paper, we study the complexity of CA through their computational capacity; namely, we study their ability to emulate one another; this notion was first introduced by Mazoyer and Rapaport (1999). We present the *emulation relation* between a pair of CA, and we demonstrate the results on a toy class of elementary CA; namely, we present their computational hierarchy. The results inspired us to define a new notion of chaos for discrete dynamical systems connected to their computational capacity.

Introducing Cellular Automata

A cellular automaton can be perceived as a k-dimensional grid consisting of identical finite state automata with the same local neighborhood. They are all updated synchronously in discrete time steps according to a fixed local update rule which is a function of the neighbors' states. A formal definition can be found in Kari (2005).

Basic Notions

We say that $\mathbb{Z}$ is a *one-dimensional cellular grid* and we call its elements the *cells*. Let S be a finite set. An S-configuration of the grid is a mapping $c : \mathbb{Z} \to S$, we write $c_i = c(i)$ for each i. We define the *nearest-neighbors relative neighborhood* of each cell $i \in \mathbb{Z}$ to be the triple $(i-1, i, i+1)$. In this paper, we will study 1-dimensional CA with nearest neighbors. Each such CA is characterized by a tuple (S, f) where S is a finite set of *states* and $f : S^3 \to S$ is a *local transition rule* of the CA. *The global rule of the CA (S, f) operating on an infinite grid* is a mapping $F : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ defined as:

$$F(c)_i = f(c_{i-1}, c_i, c_{i+1}).$$

For practical purposes, when observing the CA simulations, we consider the grid to be of finite size with a periodic boundary condition. In such a case, we compute the cells in the relative neighborhood modulo the size of the grid.

Elementary CA *Elementary cellular automata* (ECA) are 1D nearest-neighbors CA with states $S = \{0, 1\}$. We identify each local rule f determining an ECA with the *Wolfram number* of f defined as:

$$2^0 f(0,0,0) + 2^1 f(0,0,1) + 2^2 f(0,1,0) + \ldots + 2^7 f(1,1,1).$$

We will refer to each ECA as a "rule k" where k is the corresponding Wolfram number of its underlying local rule. The class of ECA is a frequently used toy model for studying different CA properties due to its relatively small size; there are only 256 of them.

For an ECA operating on a cyclic grid of size n with global rule F we define the *trajectory of a configuration* $u \in \{0, 1\}^n$ to be $(u, F(u), F^2(u), \ldots)$. The space-time diagram of such a simulation is obtained by plotting the configurations as horizontal rows of black and white squares (corresponding to states 1 and 0) with time progressing downwards.

CA Complexity via Computational Capacity

Classically, the complexity of a CA is demonstrated by its computational capacity. Intuitively, we believe that CA capable of computing non-trivial tasks should be more complex than those that are not. In the past, many different computational problems were considered, such as the majority computation task (Mitchell et al. (2000)) or the firing squad synchronization (Mazoyer (1986)), a detailed overview was written by Mitchell (1998). Nevertheless, the most classical task is the simulation of a computationally universal system (a Turing machine, a tag system, etc.). Over the years, many different CA were designed or showed to be Turing complete. However, it seems unnatural to demonstrate the complexity of an inherently parallel system by embedding a sequential computational model into it. In our opinion, an ideal set of benchmark tasks helping us determine the computational capacity of CA should

- consist of tasks suitable for the parallel computational environment
- be challenging enough
- for a given CA and a task T, it should be effectively verifiable whether the CA can compute T.

For a fixed class $\mathcal{C}$ of CA, a natural task is the following:

Given a CA in $\mathcal{C}$, how many other CA in $\mathcal{C}$ can it simulate?

The key problem is finding a suitable definition of CA "simulating" one-another. Various approaches have been suggested — simulations can be interpreted via CA coarse-grainings (Israeli and Goldenfeld (2006)), or through embedding a local rule to larger size cell-blocks (Mazoyer and Rapaport (1999), Ollinger (2001)). Many interesting theoretical results stem from such definitions. For instance, Ollinger (2001) defined a CA simulation notion which admits a universal CA — one that is able to simulate all other CA with the same dimensionality; it was designed by Culik II and Albert (1987). In this paper, we consider a notion very similar to the one suggested by Mazoyer and Rapaport (1999). Compared to their definition, ours is stricter and, thus, slightly easier to verify. We call it *CA emulation* and introduce it in the subsequent section.

CA Emulations

Basic Notions

Definition 1 (Subautomaton).
Let $\text{ca}_1 = (S, f)$, $\text{ca}_2 = (T, g)$ *be two nearest-neighbor 1D CA. We say that* ca_1 *is a subautomaton of* ca_2 *if there exists a one-to-one encoding* $e : S \to T$ *such that*

$$e(f(s_1, s_2, s_3)) = g(e(s_1), e(s_2), e(s_3))$$

for all $(s_1, s_2, s_3) \in S^3$.

This simply means that we can embed the rule table of ca_1 into the rule table of ca_2; the definition is equivalent to saying that (S, f) is a subalgebra of (T, g).

Example 2. *For an ECA* $\text{ca} = (\{0, 1\}, f)$, *we define the dual ECA* $\text{ca}' = (\{0, 1\}, f')$ *as*

$$f'(b_1, b_2, b_3) = 1 - f(1 - b_1, 1 - b_2, 1 - b_3).$$

f' simply changes the role of 0 and 1 states. The encoding $e(b) = 1 - b$, $b \in \{0, 1\}$ *witnesses that* ca *is a subautomaton of* ca' *and vice versa.*
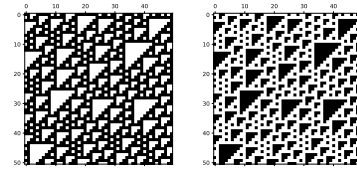

Figure 1: Space-time diagram of ECA rule 110 on the left and its dual rule 137 on the right.

Each ECA contains at most two subautomata from the ECA class: itself and its dual ECA. Hence, the subautomaton relation would not produce a rich hierarchy. It is, however, a key concept for defining the CA emulation. Below, we restrict the terminology to ECA, but the theory can be generalized to any 1D CA in a straightforward way.

We will write $\mathbf{2} = \{0, 1\}$ and define $\mathbf{2}^+ = \bigcup_{k=1}^{\infty} \mathbf{2}^k$ to be the set of all finite nonempty binary sequences.

Definition 3 (Unravelling a Boolean function).
Let $f : \mathbf{2}^3 \to \mathbf{2}$. *We define* $\widetilde{f} : \mathbf{2}^+ \to \mathbf{2}^+$ *as*

$$\widetilde{f}(b_1, b_2, \ldots, b_n) = f(b_1, b_2, b_3) \ldots f(b_{n-2}, b_{n-1}, b_n)$$

for any binary sequence $b_1 b_2 \ldots b_n$, $n \geq 3$.

By $\widetilde{f}^k$ we simply mean the composition

$$\widetilde{f}^k = \underbrace{\widetilde{f} \circ \widetilde{f} \circ \ldots \circ \widetilde{f}}_{k-\text{times}}.$$

Each iteration of $\widetilde{f}$ shortens the input size by 2, therefore we can notice that when we restrict the domain of $\widetilde{f}^k$ we get

$$\widetilde{f}^k \Big|_{\mathbf{2}^{3k}} : \mathbf{2}^k \times \mathbf{2}^k \times \mathbf{2}^k \to \mathbf{2}^k.$$

Hence, $\widetilde{f}^k$ can be interpreted as a ternary function operating on *supercells* of k bits and we obtain a CA $(\mathbf{2}^k, \widetilde{f}^k)$ whose dynamics is completely governed by the simple local rule f. Therefore, each ECA $(\mathbf{2}, f)$ gives rise to a series of CA

$$(\mathbf{2}, f), (\mathbf{2}^2, \widetilde{f}^2), (\mathbf{2}^3, \widetilde{f}^3), (\mathbf{2}^4, \widetilde{f}^4), \ldots$$

Example 4. *Suppose we have* $f : \mathbf{2}^3 \to \mathbf{2}$, $f(b_1, b_2, b_3) = b_1 \oplus b_2 \oplus b_3$, *where* $\oplus$ *is the XOR operation. In Figure 2 we show the diagram of* $\widetilde{f}^2$ *computation.*
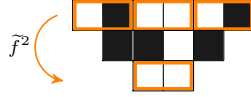
Figure 2: Diagram of $\widetilde{f}^2$ computing on supercells of size 2.

Let $e : S \to T$ be a mapping between some finite sets. We define $\overline{e} : S^+ \to T^+$ simply as

$$\overline{e}(s_1, s_2, \ldots, s_n) = e(s_1), e(s_2), \ldots, e(s_n)$$

for each $s_1, \ldots, s_n \in S$, $n \in \mathbb{N}$.

Definition 5 (ECA Emulation). *We say that* $\mathrm{ca}_1 = (\mathbf{2}, f)$ *can be emulated by* $\mathrm{ca}_2 = (\mathbf{2}, g)$ *with a supercell size k, if* ca_1 *is a subautomaton of* $(\mathbf{2}^k, \widetilde{g}^k)$. *In such a case, we write* $\mathrm{ca}_1 \leq_k \mathrm{ca}_2$.

Hence, $(\mathbf{2}, f) \leq_k (\mathbf{2}, g)$ holds if and only if there exists a one-to-one encoding

$$\mathrm{enc} : \mathbf{2} \to \mathbf{2}^k$$

such that for any initial configuration $c \in \mathbf{2}^3$ it holds that

$$\mathrm{enc}(f(c)) = \widetilde{g}^k(\overline{\mathrm{enc}}(c)).$$

We call such encoding the *witnessing encoding*. In other words, the following diagram commutes.

$$
\begin{array}{ccc}
\mathbf{2}^3 & \xrightarrow{\overline{\mathrm{enc}}} & (\mathbf{2}^k)^3 \\
\downarrow{f} & \circ & \downarrow{\widetilde{g}^k} \\
\mathbf{2} & \xrightarrow{\mathrm{enc}} & \mathbf{2}^k
\end{array}
\tag{1}
$$

We say that ca_1 can be emulated by ca_2 if there exists some k for which $\mathrm{ca}_1 \leq_k \mathrm{ca}_2$, and we write $\mathrm{ca}_1 \leq \mathrm{ca}_2$. If $\mathrm{ca}_1 \leq_1 \mathrm{ca}_2$ we say that ca_2 emulates ca_1 *trivially*. If $\mathrm{ca} \leq_k \mathrm{ca}$ for $k > 1$ we say that the ca is *self-similar*.

In the next section, we give some simple proofs of the key properties of the $\leq$ relation. Namely, we prove that $\leq$ is a preorder and that whenever $\mathrm{ca}_1 \leq \mathrm{ca}_2$ and ca_1 is Turing complete, then ca_2 is also Turing complete.

Properties of the Emulation Relation

Preservation of Turing Completeness

Lemma 6. *Suppose that* $(\mathbf{2}, f) \leq_k (\mathbf{2}, g)$ *with a witnessing encoding* enc. *Then, for each configuration* $c \in \mathbf{2}^l$, $l \geq 3$, *it holds that* $\overline{\mathrm{enc}}(\widetilde{f}(c)) = \widetilde{g}^k(\overline{\mathrm{enc}}(c))$.

Proof. Let $c \in \mathbf{2}^l$ for some $l \geq 3$. Then, $\widetilde{f}(c) \in \mathbf{2}^{l-2}$, and $\overline{\mathrm{enc}}(\widetilde{f}(c))$ is a sequence of length $l - 2$, each of its elements being a binary k-tuple. Below, we show that for each $1 \leq i \leq l - 2$ it holds that $\overline{\mathrm{enc}}(\widetilde{f}(c))_i = \widetilde{g}^k(\overline{\mathrm{enc}}(c))_i$.

$$
\begin{aligned}
\overline{\mathrm{enc}}(\widetilde{f}(c))_i &= \mathrm{enc}(\widetilde{f}(c)_i) \\
&= \mathrm{enc}(f(c_{i-1}, c_i, c_{i+1})) \\
&= \widetilde{g}^k(\mathrm{enc}(c_{i-1}), \mathrm{enc}(c_i), \mathrm{enc}(c_{i+1})) \\
&= \widetilde{g}^k(\overline{\mathrm{enc}}(c))_i.
\end{aligned}
$$

$\square$

Using the previous result it can easily be shown that for each $t \in \mathbb{N}$ and for each $c \in \mathbf{2}^+$ sufficiently long it holds that:

$$\overline{\mathrm{enc}}(\widetilde{f}^t(c)) = \widetilde{g}^{kt}(\overline{\mathrm{enc}}(c)) \tag{2}$$

simply using induction on t.

Therefore, if $(\mathbf{2}, f) \leq_k (\mathbf{2}, g)$ then the diagram (1) commutes for arbitrarily long configurations and for arbitrarily many iterations of the functions $\widetilde{f}$ and $\widetilde{g}^k$. Therefore, any space-time diagram produced by $\mathrm{ca}_1 = (\mathbf{2}, f)$ can be efficiently encoded by enc to a space-time diagram of $\mathrm{ca}_2 = (\mathbf{2}, g)$. Hence, if ca_1 is Turing complete, ca_2 must be also.

Emulation Relation Is a Preorder Clearly, for each ca it holds that $\mathrm{ca} \leq_1 \mathrm{ca}$. Hence, $\leq$ is reflexive.

Lemma 7. *The relation* $\leq$ *is transitive.*

Proof. Suppose that $(\mathbf{2}, f) \leq_k (\mathbf{2}, g)$ with witnessing encoding φ and $(\mathbf{2}, g) \leq_l (\mathbf{2}, h)$ with witnessing encoding ψ for some $k, l \in \mathbb{N}$. We define $\mathrm{enc} = \overline{\psi} \circ \varphi : \mathbf{2} \to \mathbf{2}^{kl}$. Then, for any configuration $c \in \mathbf{2}^3$ we have that:

$$
\begin{aligned}
\mathrm{enc}(f(c)) &= \overline{\psi}(\varphi(f(c))) \\
&= \overline{\psi}(\widetilde{g}^k(\overline{\varphi}(c))) \\
&= \widetilde{h}^{kl}(\overline{\psi} \circ \overline{\varphi}(c)) \\
&= \widetilde{h}^{kl}(\overline{\mathrm{enc}}(c)).
\end{aligned}
$$

In the third equality we use the result from (2). $\square$

Hence, $\leq$ is transitive and therefore, a preorder.

In contrast, Mazoyer and Rapaport (1999) define that $(\mathbf{2}, f)$ can be emulated by $(\mathbf{2}, g)$ if there exist $k, l \in \mathbb{N}$ such that $(\mathbf{2}^k, \widetilde{f}^k)$ is a subautomaton of $(\mathbf{2}^l, \widetilde{g}^l)$. They present

much deeper theoretical results about the notion in their paper. In contrast, we concentrate on explicitly computing which ECA can emulate which to form their computational hierarchy and discuss the results.

Emulations as Subalgebras   For each ECA $(\mathbf{2}, g)$ we have a sequence of algebras:

$$(\mathbf{2}, g), (\mathbf{2}^2, \widetilde{g}^2), (\mathbf{2}^3, \widetilde{g}^3), (\mathbf{2}^4, \widetilde{g}^4), \dots$$

It holds that $(\mathbf{2}, f) \leq_k (\mathbf{2}, g)$ if and only if $(\mathbf{2}^k, \widetilde{g}^k)$ contains a two-element subalgebra isomorphic to $(\mathbf{2}, f)$; the witnessing encoding being the corresponding algebra homomorphism.

Israeli and Goldenfeld (2006) have defined a different notion of CA simulation called the CA coarse-graining. It is interesting to notice that their notion is dual to the CA emulation we have defined in this paper. More specifically, the CA coarse-graining directly corresponds to the congruences of algebras. Namely, $(\mathbf{2}, g)$ can be coarse-grained into $(\mathbf{2}, f)$ if and only if there exists some $k \in \mathbb{N}$ such that $(\mathbf{2}, f)$ is isomorphic to some quotient algebra of $(\mathbf{2}^k, \widetilde{g}^k)$.

The interpretation of CA emulation as subalgebras comes in handy when developing an effective algorithm for computing the $\leq_k$ for a given supercell size k.

Emulation Computing Algorithm

Checking whether $\text{ca}_1 \leq_k \text{ca}_2$ gets infeasible for large k. In this section, we present an algorithm computing the $\leq_k$ relation for a given k and compare it to the "naive" algorithm in terms of efficiency.

Algorithm 1: Naive algorithm checking whether $\text{ca}_1 \leq_k \text{ca}_2$.

Input : $\text{ca}_1 = (\mathbf{2}, f)$, $\text{ca}_2 = (\mathbf{2}, g)$, supercell size k
Output: a witnessing enc if $\text{ca}_1 \leq_k \text{ca}_2$; "cannot emulate" message otherwise

1 **for** $\text{enc}(0), \text{enc}(1) \in \mathbf{2}^k$
2 **do**
3   encoding is valid;
4   **for** $i, j, k \in \mathbf{2}$ **do**
5   **if** $\text{enc}(f(i, j, k)) \neq \widetilde{g}^k(\overline{\text{enc}}(i, j, k))$ **then**
6   encoding is not valid;
7   break
8   **end**
9   **end**
10   **if** *encoding is valid* **then**
11   return enc
12   **end**
13 **end**
14 return "cannot emulate";

The naive algorithm simply goes through all possible encodings to determine whether $\text{ca}_1 \leq_k \text{ca}_2$. The subalgebra algorithm computes all two-element subalgebras of $(\mathbf{2}^k, \widetilde{g}^k)$ and thus, determines all ca_1 for which $\text{ca}_1 \leq \text{ca}_2$.

Algorithm 2: Subalgebra algorithm computing all ca_1 for which $\text{ca}_1 \leq_k \text{ca}_2$.

Input : $\text{ca}_2 = (\mathbf{2}, g)$, supercell size k
Output: all $\text{ca}_1 = (\mathbf{2}, f)$ together with a witnessing enc such that $\text{ca}_1 \leq_k \text{ca}_2$

1 **for** $u, v \in \mathbf{2}^k$
2 **do**
3     $\{u, v\}$ is a valid sublagebra of $(\mathbf{2}^k, \widetilde{g}^k)$;
4   **for** $i, j, k \in \{u, v\}$ **do**
5   **if** $\widetilde{g}^k(i, j, k) \notin \{u, v\}$ **then**
6             $\{u, v\}$ is not a valid subalgebra;
7   break
8   **end**
9   **end**
10   **if** $\{u, v\}$ *is a valid sublagebra* **then**
11   put $\text{enc}(0) = u$, $\text{enc}(1) = v$;
12   determine f for which $f = \text{enc}^{-1} \circ \widetilde{g}^k \circ \overline{\text{enc}}$;
13   save (f, enc)
14   **end**
15 **end**

The time complexity with respect to the supercell size k is in $\mathcal{O}(k \cdot 2^{2k})$ for both algorithms. However, Algorithm 1 needs to iterate over all ECA to compute the same result as Algorithm 2. Experimentally, we have indeed observed that Algorithm 1 takes approximately 250 times longer than Algorithm 2 to compute the emulated automata.

Emulation Hierarchy of ECA

In this section, we present a computational hierarchy based on the emulation relation. We were able to compute $\leq_k$ for supercell size k ranging from 1 to 11 due to the computational limitations.

In Example 2 we have seen that for $\text{ca} = (\mathbf{2}, f)$ and its dual $\text{ca}' = (\mathbf{2}, f')$ it holds that $\text{ca} \leq_1 \text{ca}'$ and $\text{ca}' \leq_1 \text{ca}$. Thus, $\leq_1$ is an equivalence relation on the set of ECA, each class containing exactly f and its dual f' (those might coincide for some rules). From the transitivity of $\leq$ we know that ca can emulate (and be emulated by) exactly the same rules as ca'. Using a simple program, we obtained 136 different ECA equivalence classes given by $\leq_1$. In the following text, we will identify each equivalence class with the ECA it contains having the smaller Wolfram number. We show the hierarchy for such representatives and for supercells of size k ranging from 2 to 11. Specifically, whenever we found that $\text{ca}_1 \leq_k \text{ca}_2$ for some $k \in \{2, 3, \dots, 11\}$ we represent it by the following diagram.

Many ECA are capable of emulating simple rules, such as rule 0 or the identity rule 204. Adding so many edges to the diagram would make it unreadable. Hence, we present the hierarchy in three parts.
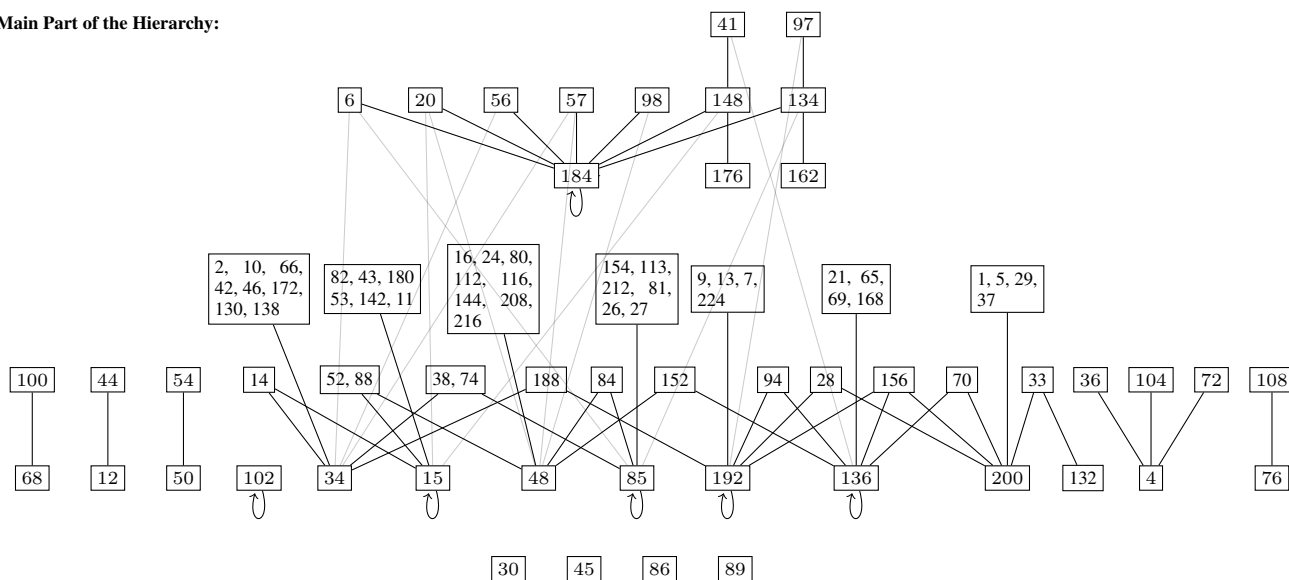
Main Part of the Hierarchy:

Figure 3: Emulation Hierarchy of ECA computed for supercell sizes ranging from 2 to 11; main part of the diagram. Some edges are depicted in light gray purely for better understandability. A looped arrow marks self-similar rules.
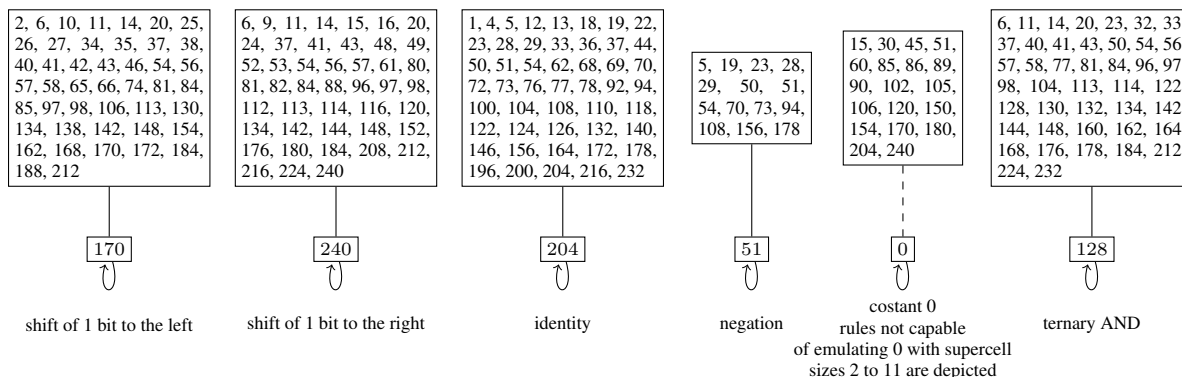
Frequently Emulated Rules:

Figure 4: Emulation Hierarchy of ECA computed for supercell sizes ranging from 2 to 11, this part shows the most frequently emulated rules.
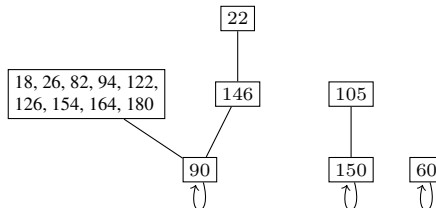
Emulated Linear Rules:

Figure 5: Emulation Hierarchy of ECA computed for supercell sizes ranging from 2 to 11, this part shows rules emulating particular linear ECA.

Main Part   For compactness, we often show multiple rules in the same node in Figure 3. However, that does not imply that such rules can emulate one another. Rules in one node are emulated and can emulate exactly the same rules contained in the main part in Figure 3. However, they do not necessarily emulate the same trivial or linear rules in Figures 4 and 5. Therefore, we note that rules in the same node do not necessarily have an identical computational capacity.

A task frequently studied in the CA environment is to determine the majority of 0's and 1's in the input configuration. The strict version of the task requires the CA to enter a homogenous state of all 0's or all 1's to indicate the result. It has been shown that no ECA can solve this strict version. If we relax the requirements on the form of the output, Capcarrere et al. (1996) have shown that rule 184 solves the majority task exactly. From the main part of the hierarchy, we can see that by encoding the input configuration in a simple way, at least nine more ECA and their dual rules can solve the task.
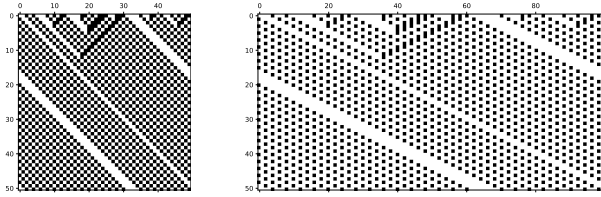

Figure 6: Space-time diagram of ECA rule 184 on the left. The emulated computation by rule 148 showed on the right. The emulation uses supercells of size 2 and every second time step is depicted.

Frequently Emulated Rules   The CA emulation offers a natural definition of a CA supporting memory: we say that a CA has a *capability of perfect memory* if it can emulate the identity rule 204 (alternatively, we could also tolerate the emulation of shifting rules or the negation). Hence, we have a practical criterion for checking whether a given CA can support memory effectively, i.e., with a small supercell size. Rules, which cannot emulate either of the rules 51, 204, 170, and 240 with the examined supercell sizes are: 0, 3, 7, 8, 17, 21, 22, 30, 32, 45, 60, 64, 86, 89, 90, 102, 105, 128, 136, 150, 160, 192. We can notice that they include trivial rules (0, 3, 8, 128), linear rules (60, 90, 105, 150), as well as the chaotic rules (22, 30, 45, 86, 89).

Emulating Linear Rules   We say an ECA is linear, if its local rule f is a linear Boolean function; i. e., it satisfies $f(x + y) = f(x) + f(y)$ for all $x, y \in \mathbf{2}^3$. Such ECA can be studied algebraically which lead to exact results describing e.g., their transients or the structure of attractors (Martin et al. (1984), Jen (1988)).

Jen (1999) has shown that the nonlinear rules 18, 126, and 146 can be mapped onto the linear rule 90. Figure 5 agrees with the results and further shows other nonlinear rules with this property which makes them interesting candidates for further algebraic studies.

Bottom Part of the Hierarchy

Top and Bottom of the Emulation Hierarchy		
ECA rules	Number of rules they can emulate	
41, 97	9	
6, 20, 54, 57, 134, 148	7	Top
14, 37, 56, 84, 94, 98, 156	6	
0, 3, 8, 17, 60, 64, 90, 102, 105, 106, 120, 150, 170, 204, 240	1	Bottom
30, 45, 86, 89	0	

Table 1: ECA Rules at the Top and Bottom of the Emulation Hierarchy computed for supercells of size 1 to 11.

There are exactly four rules (and their duals) that seem to be unable to emulate any ECA non-trivially (i.e., with a supercell size larger than one). Rule 86 is obtained from rule 30 by changing the role of the "left and right neighbor". Rule 89 is obtained in the same way from the dual of rule 45. All the four rules seem to belong to the most disordered ECA according to metrics such as the compression size of space-time diagrams (Zenil (2009)) or the transient classification (Hudcová and Mikolov (2020)). Due to the seemingly random patterns it produces, rule 30 was implemented as a pseudorandom number generator in Mathematica.

It is interesting that the most chaotic ECA seem to be exactly those unable to emulate any ECA non-trivially. To further explore this, we asked whether the four chaotic CA can emulate *any* 1D nearest-neighbors CA non-trivially, not just the ECA.

Let (S, f) be a CA. Since S is finite, there must exist some $s \in S$ and $k \in \mathbb{N}$ such that $\widetilde{f}^k(s, s, s, \ldots, s) = s$. Hence, for any CA with one state $(\{q\}, g)$ the encoding $\mathrm{enc} : q \mapsto (\underbrace{s, s, \ldots, s}_{k-\text{times}})$ witnesses that (S, f) can emulate $(\{q\}, g)$. Therefore, any CA can emulate every CA with one state. The question is whether the four CA can emulate any CA with more than one state non-trivially.

We have examined the sequence of algebras

$$(\mathbf{2}, f), (\mathbf{2}^2, \widetilde{f}^2), (\mathbf{2}^3, \widetilde{f}^3), (\mathbf{2}^4, \widetilde{f}^4), \ldots, (\mathbf{2}^{11}, \widetilde{f}^{11})$$

up to supercell size 11 for rules 30, 45, 86, and 89, and computed all their subalgebras, not only the two-element ones. We found out that all of the four CA only contain subalgebras with one element, hence, they cannot emulate any CA with more than one state for supercell sizes 2 to 11.

It remains an open problem whether this would hold for supercells of arbitrary size. However, we can conclude that if any of the four chaotic ECA can emulate any non-trivial

automaton, it cannot do so effectively, i.e., with a supercell of small size.

We note that analogous experiments were conducted for the coarse-graining relation. (Dzwinel and Magiera (2015)) showed that the rules 30 and 45, together with their negations, seem to be exactly those that cannot be coarse-grained non-trivially. These results motivate us to study chaos from a computational perspective in the next section.

Chaos In Cellular Automata

Intuitively, a chaotic CA has unpredictable dynamics and seemingly random space-time diagrams with no apparent higher-order structures forming. Below, we discuss examples of formal definitions of chaos suggested in the past and propose a new one based on the CA emulation notion.

For discrete systems, chaos is classically studied through topological dynamics, and it is typically connected to the system's sensitivity to initial conditions (Devaney (1989), Cattaneo et al. (1999)). Other studies of chaos in CA examine e.g., their input entropy (Wuensche (1999)) or properties of the local rule itself (Wuensche (2009)); a comprehensive study of ECA, including their chaotic behavior, was introduced by Chua et al. (2007).

From a great review by Martinez (2013) one can see that some definitions of chaos admit either the shift CA (Devaney (1989)) or linear CA (Cattaneo et al. (1999)) to be chaotic. In the first case, shifting a configuration by one bit does not intuitively feel very unpredictable. For linear automata, it is known that they can be simulated on a computer significantly faster than general CA (Robinson (1987)). Hence, their dynamics can be predicted quite efficiently. Below, we propose a new, much stricter definition of chaotic behavior.

Definition 8. *We call a 1D nearest-neighbors CA computationally chaotic if it cannot emulate any 1D nearest-neighbors CA with more than one state non-trivially.*

Proving that a particular CA is computationally chaotic would, in principle, require verifying an infinite amount of conditions and might require deep theoretical insight into the dynamics of the CA. However, it is useful as a new theoretical notion, which formalizes the unpredictability of a CA. Indeed, if $ca_1 \leq_k ca_2$ then for a subset of initial conditions, the result of ca_2 run for k time-steps is equivalent to simply running ca_1 for 1 time-step. Hence, at least some part of its dynamics can be predicted more effectively. In contrast, computationally chaotic CA cannot contain any simpler CA as a substructure in its space-time dynamics. This suggests that no part of its dynamics can be predicted efficiently. Hence, the definition seems to agree with the intuitive understanding of unpredictability.

In CA with trivial dynamics, structures tend to die out quite quickly. This enables such rules to emulate either the rule 0 non-trivially (this can be seen from Figure 4) or to be self-similar (e.g., rules 0, 240, or the shift rule). Thus,

they are not computationally chaotic. As linear CA are self-similar, it follows that they are not computationally chaotic either.

From the results we presented, it follows that the only ECA that could be computationally chaotic are rules 30 and 45, together with their symmetrical rules. Hence, we can form the following hypothesis.

Hypothesis 9. *ECA rules 30 and 45 are computationally chaotic.*

Proving this result would imply that such rules cannot compute any task in the sense of CA emulation.

We note that the definition can be extended to CA in arbitrary dimensions with various neighborhood shapes in a straightforward way. We also note that the definition does not depend on whether the CA operates on a finite or infinite grid and is purely a property of its local rule.

Most definitions of chaos in discrete systems are not related to their computational capacity. Hence, there is an interesting ongoing debate whether chaotic CA can compute any non-trivial tasks. In contrast, the notion of computational chaos is strongly connected to the computational capacity of the CA. If a CA is computationally chaotic, it cannot compute the dynamics of any other CA. Hence, we might conclude that being computationally chaotic directly implies the inability to compute a non-trivial task.

Conclusion

We studied the notion of CA emulation as a method of determining the computational capacity of CA. We showed that the CA emulation relation forms a preorder on the ECA class and presented an approximation of the emulation hierarchy produced by the preorder. We did notice that the most chaotic CA seem to be the minimal elements of the hierarchy. This inspired us to introduce a new definition of chaos in the CA environment. Our notion of chaos is novel because it is connected to the computational capacity of a system. In contrast to previous concepts of chaos, our definition does not regard linear and shifting automata as chaotic. This agrees with the results that the dynamics of such CA can be predicted more efficiently than for general CA.

The emulation relation can be defined for CA in any dimension and with an arbitrary neighborhood. Though its computation requires verifying infinitely many conditions, we can compute it just for supercells of small size. Verifying such small supercell values helps us determine whether a CA can emulate any others effectively.

A pair of ECA obtained by changing the role of their "left and right neighbor" has equivalent dynamics; however, the emulation relation we presented is unable to discover such an equivalence. For this reason, it is meaningful to study other possible definitions of a CA subautomaton. It would be very interesting to examine whether the different variants of the definition would change the computational hierarchy results significantly.

As a possible application, we can use the method of CA emulation to construct Turing-complete CA without having to give elaborate proofs of this fact. We would simply embed a well-known Turing-complete CA into a newly constructed CA with possibly much richer dynamics. It could be interesting to design a CA capable of emulating many different CA and study the dynamics of such a rich system.

Acknowledgements

References

Capcarrere, M. S., Sipper, M., and Tomassini, M. (1996). Two-state, $r = 1$ cellular automaton that classifies density. *Phys. Rev. Lett.*, 77:4969–4971.

Cattaneo, G., Formenti, E., Margara, L., and Mauri, G. (1999). On the dynamical behavior of chaotic cellular automata. *Theoretical Computer Science*, 217(1):31–51.

Chua, L. O., Guan, J., Sbitnev, V. I., and Shin, J. (2007). A nonlinear dynamics perspective of wolfram's new kind of science. part vii: Isles of eden. *International Journal of Bifurcation and Chaos*, 17(09):2839–3012.

Cook, M. (2004). Universality in elementary cellular automata. *Complex Systems*, 15.

Culik II, K. and Albert, J. (1987). A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1:1–16.

Devaney, R. L. (1989). *An Introduction To Chaotic Dynamical Systems, Second Edition*. Addison-Wesley advanced book program. Avalon Publishing.

Dzwinel, W. and Magiera, K. (2015). Irreducible elementary cellular automata found. *Journal of Computational Science*, 11:300–308.

Gutowitz, H. A. (1990). A hierarchical classification of cellular automata. *Physica D: Nonlinear Phenomena*, 45(1):136–156.

Hudcová, B. and Mikolov, T. (2020). Classification of complex systems based on transients. *Artificial Life Conference Proceedings*, (32):367–375.

Israeli, N. and Goldenfeld, N. (2006). Coarse-graining of cellular automata, emergence, and the predictability of complex systems. *Physical Review E*, 73(2).

Jen, E. (1988). Linear cellular automata and recurring sequences in finite fields. *Communications in Mathematical Physics*, 119:13–28.

Jen, E. (1999). Exact solvability and quasiperiodicity of one-dimensional cellular automata. *Nonlinearity*, 4:251.

Kari, J. (2005). Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1):3 – 33.

Kůrka, P. (2009). Topological dynamics of cellular automata. *Cellular Automata. Encyclopedia of Complexity and Systems Science Series*.

Martin, O., Odlyzko, A., and Wolfram, S. (1984). Algebraic properties of cellular automata. *Communications in Mathematical Physics*, 93.

Martinez, G. J. (2013). A note on elementary cellular automata classification. *Journal of cellular automata*, 8.

Martinez, G. J., Seck Tuoh Mora, J., and Zenil, H. (2013). Computation and universality: Class iv versus class iii cellular automata. *Journal of cellular automata*.

Mazoyer, J. (1986). An overview of the firing squad synchronization problem. *Automata Networks, LITP Spring School on Theoretical Computer Science Proceedings*, 316.

Mazoyer, J. and Rapaport, I. (1999). Inducing an order on cellular automata by a grouping operation. *Discrete Applied Mathematics*, 91(1):177–196.

Mitchell, M. (1998). Computation in cellular automata: A selected review. *Non-Standard Computation*, pages 95–140.

Mitchell, M., Crutchfield, J., and Das, R. (2000). Evolving cellular automata with genetic algorithms: A review of recent work. *First Int. Conf. on Evolutionary Computation and Its Applications*, 1.

Ollinger, N. (2001). Toward an algorithmic classification of cellular automata. *LIP 222001-10*, (2).

Robinson, A. (1987). Fast computation of additive cellular automata. *Complex Systems*, 1(1):211–216.

Toffoli, T. (1977). Computation and construction universality of reversible cellular automata. *Journal of Computer and System Sciences*, 15(2):213 – 231.

Wolfram, S. (1984). Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1):1–35.

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

Wuensche, A. (1999). Classifying cellular automata automatically: Finding gliders, filtering, and relating space-time patterns, attractor basins, and thez parameter. *Complexity*, 4:47–66.

Wuensche, A. (2009). Cellular automata encryption: the reverse algorithm, z-parameter and chain-rules. *Parallel Processing Letters*, 19:283–297.

Wuensche, A. and Lesser, M. (2001). The global dynamics of celullar automata: An atlas of basin of attraction fields of one-dimensional cellular automata. *J. Artificial Societies and Social Simulation*, 4.

Zenil, H. (2009). Compression-based investigation of the dynamical properties of cellular automata and other systems. *Computing Research Repository - CORR*, 19.