Universality Frontier for Asynchronous Cellular Automata

Ivan Baburin
Department of Computer Science, ETH Zürich, Switzerland

Matthew Cook
Faculty of Science and Engineering, University of Groningen, The Netherlands

Florian Grötschla
Department of Electrical Engineering, ETH Zürich, Switzerland

Andreas Plesner
Department of Electrical Engineering, ETH Zürich, Switzerland

Roger Wattenhofer
Department of Electrical Engineering, ETH Zürich, Switzerland

Abstract

In this work, we investigate computational aspects of asynchronous cellular automata (ACAs), a modification of cellular automata in which cells update independently, following an asynchronous update schedule. We introduce flip automata networks (FANs), a simple modification of automata networks that remain robust under any asynchronous updating order. By using FANs as a middleman, we show that asynchronous automata can efficiently simulate their synchronous counterparts with a linear memory overhead, which improves upon the previously established quadratic bound. Additionally, we address the universality gap for (a)synchronous cellular automata–the boundary separating universal and non-universal automata, which is still not fully understood. We tighten this boundary by proving that all one-way asynchronous automata lack universal computational power. Conversely, we establish the existence of a universal asynchronous 6-state first-neighbor automaton in one dimension and a 3-state von Neumann automaton in two dimensions, which represent the smallest known universal constructions to date.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Universality, Asynchronous Cellular Automata, Automata Networks

Digital Object Identifier 10.4230/LIPIcs.MFCS.2025.11

Related Version *Full version with appendix*: `https://arxiv.org/abs/2502.05989` [2]

1 Introduction

Perhaps the most famous example of a complex system is a cellular automaton (CA): a uniform array of infinitely many simple machines–cells–all behaving in the same manner and interacting using basic, entirely local rules. Somewhat paradoxically, the simplicity of local components often results in the formation of sophisticated global patterns, highlighting the elementary non-continuity underlying the very concept of complexity. A cellular automaton can be seen as complex if any given computation can be embedded into its evolution, a phenomenon known as computational universality. During the last decades, many astonishingly simple universal cellular automata were constructed [29], which, as a byproduct, resulted in the discovery of many novel universal models of computation [4, 10, 31]. On the other hand, cellular automata were also used as abstractions for biological and physical models of complexity: Game of Life is a famous example of an artificial biological system [12] while Wolfram [37] used CAs to model physical laws using local discrete processes.

© Ivan Baburin, Matthew Cook, Florian Grötschla, Andreas Plesner, and Roger Wattenhofer;
licensed under Creative Commons License CC-BY 4.0

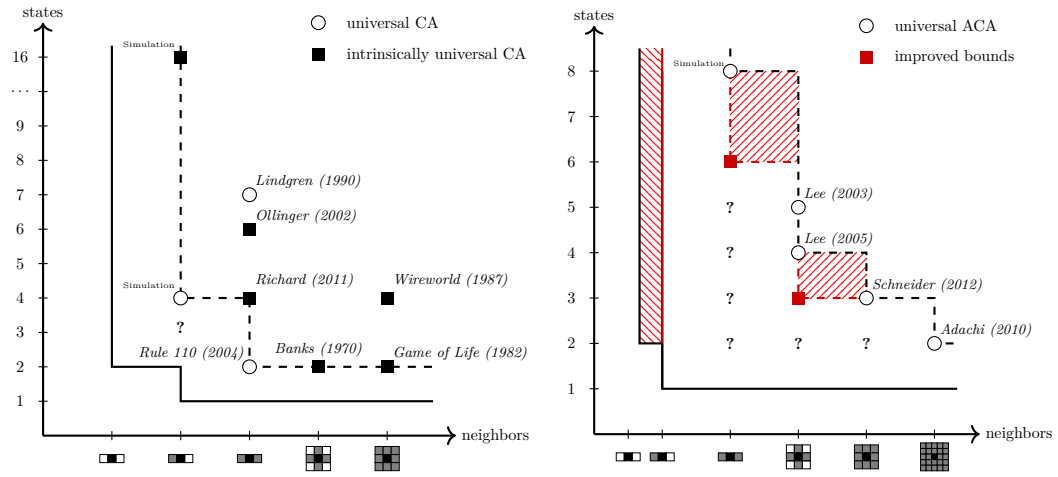

Figure 1 Current "universality frontier" for synchronous (left) and asynchronous cellular automata (right). The solid line indicates the impossibility proofs for universality, while the dashed line shows the currently smallest known universal constructions. Contributions of this work are marked in red: 6-state first neighbors, 3-state von Neumann, and non-universality of one-way automata.

Despite their mathematical beauty, cellular automata have a severe limitation when it comes to modeling the real world–the requirement of global synchronization. To mitigate this issue, asynchronous cellular automata (ACAs) have been proposed, where every cell performs interactions at its local speed, without the need for global synchronization. Although at first glance it may seem that asynchronous automata are very different from their synchronous peers, in reality, they share many similarities. Nakamura [25] has shown that for any synchronous CA, we can construct an ACA which, in a very natural way, simulates the synchronous CA. However, this simulation comes at the price of a quadratic memory overhead, with further optimizations introduced later [21]. A unified, algebraic approach to asynchrony was later given by Gács [18], who characterized a large class of cellular automata whose dynamics remains invariant with respect to any choice of updating.

Asynchronous updating allows more realistic modeling of many natural phenomena, and many potential use cases in distributed computing have been proposed [7]. For example, one can draw parallels to emergent swarm intelligence in biological systems, such as ant colonies, which use chaotic cooperation of many simple agents to solve increasingly complex transport problems [13, 9]. The implementation of asynchrony can also directly affect computational ability–for example, in chemical reaction networks, depending on the asynchronous setting, the computational power varies between primitive recursive functions, Boolean circuits, and Turing machines [5]. Similarly, although cellular automata are capable of universal computation, a certain threshold in terms of their size and state count must be met to enable it. This question has been extensively investigated for CAs [27, 23, 4, 6, 17, 3, 10, 24, 37] and ACAs [8, 20, 19, 32, 1]. Despite this, the limits of asynchronous universality are not yet known, and the smallest constructions still lag behind their synchronous counterparts, hinting at a foundational discrepancy between synchronous and asynchronous computation.

Our contributions. We present a comprehensive overview of computational abilities of (a)synchronous cellular automata and propose a formal definition for the notion of simulation with ACAs, which, although often used implicitly, has been missing in the literature [25, 21, 38, 16]. We adopt the "adversarial" view on asynchrony, i.e. require the ACAs to be

robust against *any* asynchronous updating (other notions such as randomized and "best-case" asynchrony have also been studied [7]). By combining various simulation techniques and known universality constructions, we assemble an explicit "universality frontier" for CAs and ACAs, i.e., the boundary between universal and non-universal computation, similarly to how it was already done for Turing machines [26] (see Figure 1). The dashed line represents the best-known upper bound on universality constructions, while the filled line represents the best-known lower bound for non-universality constructions. We further present our own contributions to both asynchronous simulation and the "universality frontier":

- In Section 3, we introduce a simple asynchronous model of computation–the flip automata network (FAN), which have invariant computations under any asynchronous setting.
- In Section 4 we show that by using FANs as a middleman, ACAs can simulate their synchronous counterparts with just $4q$ states (and $3q$ in the 1-dimensional case), improving on the previous constructions which had quadratic overhead.
- In Section 6 we extend the "universality frontier" by deriving a non-universality result for one-way asynchronous automata, demonstrating their computational equivalence to classical finite-state machines. This suggests that bidirectional communication is crucial to achieving universality in ACAs (which is not the case for classical CAs).
- In Section 7 we present smaller universal ACAs than previously known: a universal 3-state von Neumann automaton and a universal 6-state first neighbors automaton.

2 Background

▶ **Definition 1** (Cellular Automaton). *A d-dimensional cellular automaton (CA) is a tuple $(S, N, f, \mathbb{Z}^d)$ where $f\colon S^k \longrightarrow S$ is the local rule function, S is the set of states, $N = (n_1, n_2, \ldots, n_k)$ with $n_i \in \mathbb{Z}^d$ is the (local) neighborhood vector (defines neighbor offsets for each cell) and $\mathbb{Z}^d$ is the cellular space.*

We call a neighborhood N *symmetric* if $-1 \cdot N$ is a permutation of N, and two cells $i, j \in \mathbb{Z}^d$ are *neighboring* if they are in each other's neighborhoods. There are infinitely many choices for the neighborhood vector N, but four types of neighborhoods are considered canonical and have been researched the most (note, Moore and von Neumann generalize naturally to arbitrary dimension d, and first neighbors is the 1-dimensional case for both):

One-way *First neighbors* *Von Neumann* *Moore*

A *configuration* of a d-dimensional CA is a function $c : \mathbb{Z}^d \longrightarrow S$, which assigns a state to every cell in the cellular space. At each timestamp t, every cell $i \in \mathbb{Z}^d$ synchronously applies the local rule function to its local neighborhood N, causing the complete configuration to transition to a new configuration c_{t+1}. For each cell i the next state $c_{t+1}(i)$ is defined as

$$c_{t+1}(i) = f(c_t(i + n_0), \ldots, c_t(i + n_k))$$

We can characterize the transformation between two consecutive configurations of a CA by specifying the *global transition function* $G : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$ with $G(c_k) = c_{k+1}$.

▶ **Definition 2** (Asynchronous Cellular Automaton). *An asynchronous cellular automaton (ACA) $\mathbb{A} = (S, N, f, \mathbb{Z}^d)$ is a cellular automaton that chooses at each point in time t an update set $D_t \subseteq \mathbb{Z}^d$ and applies the local rule function f only to cells in D_t.*

Definition 2 presents the most common type of asynchronous updating, also known as *purely* asynchronous. Other notions such as *α-asynchrony* (each cell updates independently with probability α) have also been considered, for a detailed survey we refer to Fatès [7].

▶ **Definition 3** (Update Schedule). *For an asynchronous cellular automaton $\mathbb{A} = (S, N, f, \mathbb{Z}^d)$ we define some asynchronous update schedule $\zeta\colon \mathbb{N} \longrightarrow \mathcal{P}(\mathbb{Z}^d)$, a function specifying the subsets of cells updated at each point t in time. ζ has the property that every $i \in S^{\mathbb{Z}^d}$ appears infinitely often in* $\operatorname{Im}\zeta$*, or, in other words, every cell is updated infinitely often.*

For a purely asynchronous setting, ζ can be arbitrary: an "adversary" can choose any ζ to update $\mathbb{A}$. Given a ζ we identify the asynchronous global transition function $G^\zeta\colon S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$ and corresponding *asynchronous space-time* $\mathrm{st}^\zeta\colon \mathbb{N} \times \mathbb{Z}^d \longrightarrow S$ mapping configurations to their evolved asynchronous counterparts; $\mathrm{st}^\zeta(0, c_0) = c_0$ and $\mathrm{st}^\zeta(t, c_0) = c_t^\zeta$ such that

$$c_{t+1}^\zeta = G^\zeta(c_t) = G_{\zeta(t)}(c_t) \quad \text{where} \quad G_D(c)(i) = \begin{cases} G(c)(i) & \text{if} \quad i \in D \\ c(i) & \text{otherwise} \end{cases}$$

Furthermore, we call a cell $i \in \mathbb{Z}^d$ *active* (on configuration c) if and only if $G(c)(i) \neq c(i)$ (note that this is independent of the update schedule).

▶ **Definition 4** (Update History). *Given an asynchronous cellular automaton $\mathbb{A}$ under some update schedule ζ, we define its update history as a sequence of configurations $h_0, h_1, \ldots$ where $h_0 = c_0$ and $h_t(i) = c_{t_i'}(i)$ with t_i' as below if such t_i' exists, and undefined otherwise.*

$$t_i' = \min\{a_t \in \mathbb{N} \mid \exists a_0 \ldots a_{t-1}\ \forall j < t\ (a_j < a_{j+1} \wedge c_{a_j}(i) \neq c_{a_{j+1}}(i))\}$$

Intuitively, the update history records the asynchronous space-time "up to value equality." We only record "true" state updates to cell i, discarding the updates that did not change the state and the time these updates occurred (in particular, the history can be finite!). We say that an ACA $\mathbb{A}$ has *invariant histories on configuration* c_0 if all update schedules initialized with c_0 result in the same update history. Further, $\mathbb{A}$ has *invariant histories* if all initial configurations have invariant histories. Such ACAs form a particularly important subclass of CAs, since they allow for reliable computations without any need for global synchronization.

Gács [18] has given an almost complete characterization of ACAs with invariant histories using only algebraic properties of monotonicity and commutativity.

▶ **Definition 5** (Monotonicity). *A global transition function G is monotonic if for any configuration c, active cell i and an update set $A \subseteq \mathbb{Z}^d \setminus \{i\}$ it holds that*

$$G_{\{i\}}(G_A(c))(i) \neq c(i).$$

An update of any other cell cannot stop cell i from updating, ensuring a "lock-free" flow.

▶ **Definition 6** (Commutativity). *A global transition function G is commutative if for any configuration c and two disjoint update sets of active cells $A, B \subseteq \mathbb{Z}^d$ it holds*

$$G_A(G_B(c)) = G_{A \cup B}(c) = G_B(G_A(c)).$$

Commutativity implies that updates of simultaneously active cells can be performed in any order. We say that the CA is *commutative* if its global transition function is commutative.

▶ **Theorem 7** (Invariance [18]). *A cellular automaton $(S, N, f, \mathbb{Z}^d)$ is commutative if and only if it has invariant histories and its global transition function G is monotonic.*

Even though commutativity does not characterize all automata with invariant histories (this question was shown to be undecidable in the same paper), it characterizes a very natural subclass thereof, since monotonicity is, in general, desirable. Furthermore, commutativity

can be verified algorithmically–due to the local structure of G it is sufficient to check the conditions in Definition 6 only for cases where A and B are singleton update sets, i.e. whether $G_{\{i\}}(G_{\{j\}}(c)) = G_{\{i,j\}}(c)$ holds for any cells $i, j \in \mathbb{Z}^d$ (see Appendix of the full version).

A generalization of commutativity for automata networks–an extension of cellular automata allowing arbitrary spaces and non-uniform local rule functions–is known, but requires more complex machinery [15]. We note that while the algebraic characterization provides a simple way of verifying for invariant histories in ACAs, it is generally hard to design commutative ACAs with desired properties without inducing a large state complexity overhead. These issues arise due to the lack of a general "constructive" understanding of such ACAs, something we attempt to mitigate in the sections below.

3 Flip Automata Networks

In this section, we present a simple asynchronous distributed model of computation called flip automata networks (FAN). While FANs are just a simple modification of automata networks, they have the advantage of having invariant histories by design, allowing us to build a simple bridge between CAs and ACAs.

▶ **Definition 8** (Flip Automata Network). *A FAN is a triple $\mathcal{A}_{flip} = (T, S, \{f_i\}_{i\in I})$ where $T = (I, E)$ is a undirected graph with some indexing $v_i \in I$ edges $E \subseteq \{\{v_i, v_j\} \mid i \neq j \wedge v_i, v_j \in I\}$ such that each edge has a direction indicator $\in \{\nearrow, \swarrow\}$, neighborhood function*

$$N(v_i) = \{v \in I \mid \{v, v_i\} \in E\}$$

which must be finite $\forall v_i \in I$, S is a state set, and $\{f_i\}_{i\in I}$ is the set of all flip functions where

$$f_i \colon S^{N(v_i)\cup\{v_i\}} \longrightarrow S \times \{\nearrow, \swarrow\}^{N(v_i)}$$

is the flip function associated with vertex v_i ($\nearrow$ "points" to the node with the larger index and $\swarrow$ to the node with the smaller index). A function f_i is only applicable if the direction indicator on all edges between $N(v_i)$ and v_i "point" towards v_i, and the application of f_i can update the state of v_i and "flip" the direction indicator on any of the edges from $N(v_i)$.

Similarly to classical automata networks, individual transition functions can be applied in some (arbitrary) order specified by a given update schedule $\zeta \colon \mathbb{N} \longrightarrow \mathcal{P}(I)$, and we define configuration of a FAN by assigning each node a state and each edge a directional indicator. FANs have an intuitive interpretation–the arrows indicate directed information sharing and every node waits for all of its neighbors to share before performing a transition. A small example is provided in Figure 2.

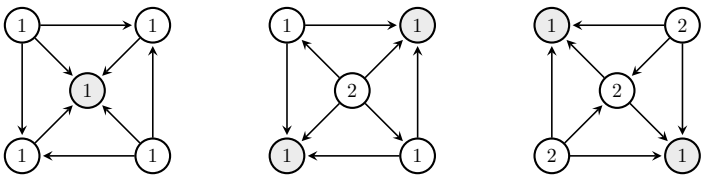

Figure 2 Two-step evolution of flip automata network $\mathcal{A}_{\text{flip}} = (T_{\text{flip}}, \{0, 1, 2\}, \{f_i\}_{i=0}^{4})$ where each f_i computes the sum of all neighboring node states mod 3, and flips all incoming arrows (applicable transitions marked in gray). At each point in time we apply all applicable transitions.

▶ **Proposition 9.** *Flip automata networks have invariant histories, i.e., the update schedule does not affect the update history of individual nodes.*

The proof for Proposition 9 can be found in Appendix of the full version. In the following sections, we demonstrate that flip automata networks can be embedded into asynchronous cellular automata with minimal overhead, enabling simpler simulation techniques and smaller universality constructions. A simple embedding is given in Example 10 below.

▶ **Example 10.** Consider an asynchronous version of Rule 212. The only transitions that affect the cell state are $(100) \mapsto 1$ and $(011) \mapsto 0$; in all other cases, the cell does not change its state. Observe that no two neighboring cells can be active at the same time. Using the local neighborhood of each cell, we can interpret Rule 212 as a 1-dimensional flip automata network. To this end we define a mapping ψ that adds directional indicators between neighboring cells i and $i+1$ based on their current states in some configuration c:

$$\psi(c(i), c(i+1)) = c(i) \longleftarrow c(i+1) \quad \text{for } c(i) = c(i+1)$$
$$\psi(c(i), c(i+1)) = c(i) \longrightarrow c(i+1) \quad \text{for } c(i) \neq c(i+1)$$

This allows us to transform any configuration of Rule 212 into a configuration of a flip automata network. For example, the following synchronous update of configuration c

$$\ldots 100110011 \ldots \mapsto \ldots 110011001 \ldots$$

can be seen as an update in the flip automata network, where all applicable transitions flip:

$$\boxed{1} \longrightarrow \boxed{0} \longleftarrow \boxed{0} \longrightarrow \boxed{1} \longleftarrow \boxed{1} \longrightarrow \boxed{0} \longleftarrow \boxed{0} \longrightarrow \boxed{1} \longleftarrow \boxed{1}$$
$$\Downarrow$$
$$\boxed{1} \longleftarrow \boxed{1} \longrightarrow \boxed{0} \longleftarrow \boxed{0} \longrightarrow \boxed{1} \longleftarrow \boxed{1} \longrightarrow \boxed{0} \longleftarrow \boxed{0} \longrightarrow \boxed{1}$$

Furthermore, if we think of $\longrightarrow$ as $\diagdown$ slope and of $\longleftarrow$ as $\diagup$ slope we can visualize any configuration of such flip automata network as a zigzag line in a 2-dimensional space, which we will refer to as a *mountain-valley* landscape, as shown in Figure 3 (left). The update history of configuration c under some asynchronous update schedule can thus be visualized as a 2-dimensional "fluid" slowly filling the space-time st, where at each point in time some subset of "valleys" become "mountains". Depending on the update schedule ζ, some mountains will grow faster than others; however, since every cell will be updated infinitely often, the "fluid" is guaranteed to fill the full space-time, as sketched in Figure 3 (right).

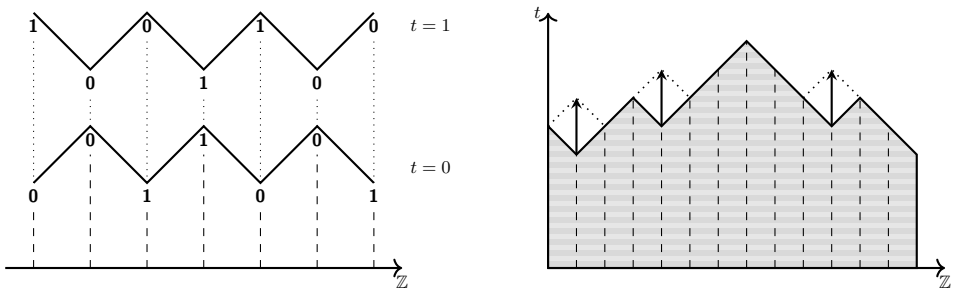

Figure 3 Representation of a flip automata network for Rule 212 as a mountain-valley landscape (left) and evolution of a mountain-valley landscape carrying information about the frequency of updates (right).

▶ **Lemma 11** (Duality). *Every ACA $\mathbb{A} = (S, N, f, \mathbb{Z}^d)$ with von Neumann neighborhood where no two neighboring cells can be simultaneously active has a "dual" FAN $\mathcal{A} = (T = \mathbb{Z}^d, S, \{f'\})$ such that every configuration of $\mathbb{A}$ can be projected to a valid configuration of $\mathcal{A}$, and for any update schedule ζ their evolutions commute with the projection.*

The proof for this result is similar to Example 10 and can be found in Appendix of the full version. As a direct consequence we obtain that all such ACAs have invariant histories due to Proposition 9 (can be alternatively shown using commutativity). Lemma 11 effectively shows the duality between a large subclass of ACAs and FANs, which will prove useful in understanding their computational mechanics.

▶ **Open question 1.** *Can duality between FANs and ACAs be extended to all commutative ACAs (perhaps with modifications to Definition 8)?*

4 Simulating Synchronous Networks

We start by discussing the notion of simulation between cellular automata and present its extension to the asynchronous scenario. For brevity, we refer to the simulated (synchronous) automaton as the "guest" CA, and the simulating (a)synchronous automaton as the "host".

▶ **Definition 12** (Direct Simulation). *A CA $B = (S^B, N^B, f^B, \mathbb{Z}^d)$ with global transition function G^B directly simulates a CA $A = (S^A, N^A, f^A, \mathbb{Z}^d)$ with global transition function G^A of the same dimension according to a mapping $\psi\colon S^A \longrightarrow \mathcal{P}(S^B) \setminus \{\varnothing\}$ if for all states $a, b \in S^A$ we have $\psi(a) \cap \psi(b) = \varnothing$ and for any configuration $c \in (S^A)^{\mathbb{Z}^d}$ we have*

$$\{G^B(c') \mid c' \in \psi(c)\} \subseteq \psi(G^A(c)) \quad \text{where} \quad \psi(c) \coloneqq i \mapsto \psi(c(i))$$

i.e. all encodings c' evolve in B as c evolves in A. In this case we write $G^A \prec G^B$.

The general definition of simulation extends the definition of direct simulation by allowing packing, cutting, and shifting, such that a direct simulation can occur.

▶ **Definition 13** (Unpacking Map). *Let S be a state set and $m = (m_1, \ldots, m_d)$ be a tuple of positive integers, then the unpacking bijective map*

$$o_m\colon \left(S^{\prod m_i}\right)^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$$

is defined for every configuration $c \in \left(S^{\prod m_i}\right)^{\mathbb{Z}^d}$ such that for every position $i \in \mathbb{Z}^d$ and offset $r \in \prod_i \mathbb{Z}_{m_i}$ it holds that $o_m(c)(m_1 i_1 + r_1, \ldots, m_d i_d + r_d) = c(i)(r)$.

In essence, the unpacking map o_m transforms every cell i with state from $S^{\prod m_i}$ into a block of cells of size $\prod m_i$ and bijectively maps the state information from i to the block.

▶ **Definition 14** (Simulation [29]). *A cellular automaton $B = (S^B, N^B, f^B, \mathbb{Z}^d)$ simulates a cellular automaton $A = (S^A, N^A, f^A, \mathbb{Z}^d)$ if for some m there exists an unpacking map o_m, a translation τ_v and a positive integer $n \in \mathbb{N}$ such that*

$$(G^A) \prec o_m^{-1} \circ (G^B)^n \circ o_m \circ \tau_v$$

In this case, we say that B simulates A in n-linear time (and in real time if $n = 1$).

For the asynchronous scenario multiple simulation definitions have been proposed, however, all of them have turned out to be either too restrictive or flawed [25, 21]. Most notably, they did not support for "transitivity", i.e., if an ACA $\mathbb{C}$ simulates a CA B, and B in turn simulates another CA A, then one would expect $\mathbb{C}$ to simulate A (which was not fulfilled). To mitigate these issues, we propose a natural extension of Definition 14 based on the notion of synchronous simulation and invariant histories, such that it is general enough to encompass most of the previous simulation constructions [25, 18] (see Appendix of the full version), while preserving the crucial "transitivity" property.

▶ **Definition 15** (Invariant Simulation). *Let $\mathbb{B} = (S^B, N^B, f^B, \mathbb{Z}^d)$ be an ACA and $A = (S^A, N^A, f^A, \mathbb{Z}^d)$ be some synchronous CA of the same dimension, and let*

$$H^B \colon (S^B)^{\mathbb{Z}^d} \longrightarrow (S^B)^{\mathbb{Z}^d}$$

be a function that transforms the configuration by mapping every cell i in state $h_t(i)$ to the next point in its invariant update history $h_{t+1}(i)$ under $\mathbb{B}$ ($H^B(c)$ is undefined in case $\mathbb{B}$ does not have invariant histories for some c). We say that $\mathbb{B}$ invariantly simulates A for some m there exists an unpacking map o_m, a translation τ_v and positive integers $k, l \in \mathbb{N}$ such that

$$(G^A)^k \prec o_m^{-1} \circ (H^B)^l \circ o_m \circ \tau_v$$

where l/k is again the linear time parameter.

Definition 15 requires configurations of A to appear as complete, encoded rows in the invariant history h of $\mathbb{B}$, allowing the observer to reconstruct the evolution of A from the evolution of $\mathbb{B}$, even if the update schedule ζ is unknown. The "transitivity" is also fulfilled, since we can chain unpackings and translations between different types of simulation.

Simulation techniques. The classical approach to asynchronous simulation requires the "host" ACA to be of the same dimension d and neighborhood N as the "guest" CA–indeed, when synchronizing a distributed system, the general goal is to alter the latter as little as possible. The classical $3q^2$ real time construction by Nakamura [25, 18] (see Appendix of the full version) and $q^2 + 2q$ state 3-linear time construction in [21, 38] achieve this by increasing the number of states in the "host" (given a q-state "guest" CA) and use one cell of the "host" to simulate one cell of the "guest". Furthermore, these approaches require the neighborhood N to be symmetric (this constraint will be investigated more in Section 5). Our new construction–based on embeddings of flip automata networks into ACAs–allows for simulations with both linear state and time overheads, which is achieved using spatial grouping of cells, as introduced in Definition 15. We also restrict the neighborhood N to be either von Neumann or Moore, since these constitute the most crucial cases.

▶ **Proposition 16** (Mountain-Valley Encoding). *For a synchronous automaton $A = (S, N, f, \mathbb{Z}^d)$ with von Neumann or Moore's neighborhood N we can construct an asynchronous cellular automaton $\mathbb{B} = (S \times \{0, 1, 2, 3\}, N, f', \mathbb{Z}^d)$ which simulates A in real time.*

Proof. The high-level idea behind the construction of $\mathbb{B}$ is that we first construct a flip automata network $\mathcal{A}$ which simulates A, and then create an asynchronous cellular automaton embedding $\mathcal{A}$. To keep the construction elegant, we present it for the 2-dimensional von Neumann neighborhood $N = ((-1, 0), (1, 0), (0, 0), (0, 1), (0, -1))$ in detail, and give a sketch for Moore's neighborhood. The generalization to any dimension d is straightforward.

Consider a FAN $\mathcal{A} = (T = \mathbb{Z}^2, S, \{f_{\mathbf{E}}, f_{\mathbf{O}}\})$ with the same set of states as A, where we distinguish between two types of nodes $\mathbf{E}$ and $\mathbf{O}$, and set $\oplus/\ominus$ to be addition and subtraction mod $|S|$. Then, for von Neumann neighborhood:

- $\mathbf{E}$ nodes are placed on coordinates $(2i, 2j) \in \mathbb{Z}^2$ and we set

$$f_{\mathbf{E}}(s_{(1,0)}, s_{(0,1)}, s, s_{(-1,0)}, s_{(0,-1)}) = f(s_{(1,0)} \ominus s, s_{(0,1)} \ominus s, s, s_{(-1,0)} \ominus s, s_{(0,-1)} \ominus s)$$

  Intuitively, $\mathbf{E}$ nodes store the states of the cells in A and simulate the local transition f
- $\mathbf{O}$ nodes are placed at all remaining coordinates and every $f_{\mathbf{O}}$ computes the sum $\oplus$ of all of its $\mathbf{E}$ neighbors. On a high level, $\mathbf{O}$ nodes use addition mod $|S|$ to accumulate and pass on the information about (simulated) neighbors of $\mathbf{E}$ nodes

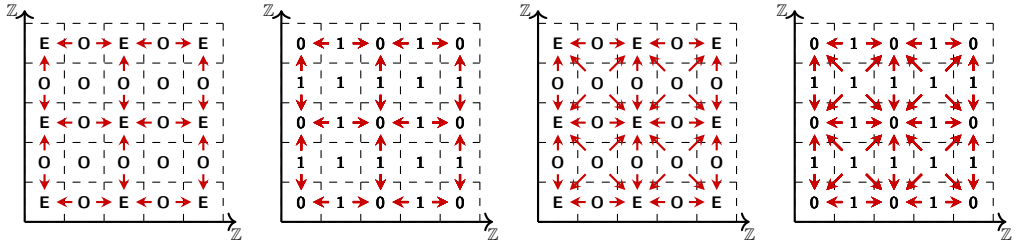

Figure 4 A FAN $\mathcal{A}$ with two types of nodes **E**, **O** and the encoding of $\mathcal{A}$ inside an ACA $\mathbb{B}$ using an even/odd timer $t \in \{0,1,2,3\}$ with von Neumann neighborhood (left) and a similar FAN with identical encoding using Moore's neighborhood (right).

Observe the topology of $\mathcal{A}$ as presented in Figure 4–the directional indicators are only added between neighboring **E** and **O** nodes. For von Neumann neighborhood (left), the **O** nodes with odd coordinates will be isolated in $\mathcal{A}$ since they do not have any **E** neighbors. On the other hand, in the case of Moore's neighborhood (right), the topology of $\mathcal{A}$ becomes non-uniform with different **O** nodes having different neighborhoods. We initialize $\mathcal{A}$ with all directional indicators pointing from **E** nodes to **O** nodes, and on every local transition, we flip all edges. Furthermore, the initial configuration c_0 of A is encoded on the **E** nodes, where $T(2i) = c_0(i)$, and **O** nodes are initialized arbitrarily. If we first update all **O** nodes, flip the arrows, and update all **E** nodes, the state of the network will be an encoding of $G^A(c_0)$.

Finally, let us construct an asynchronous cellular automaton $\mathbb{B} = (S \times \{0,1,2,3\}, N, f', \mathbb{Z}^2)$ corresponding to FAN $\mathcal{A}$. First, let us define the initial configuration c_0' on $\mathbb{B}$. For each **E** node at (i,j) we set $c_0'(2i,2j) = c_0(i,j)2$ (tuple notation), and each **O** node is initialized with $s1$ where $s \in S$ arbitrary (see Figure 4). We encode directional indicators of $\mathcal{A}$ in $\mathbb{B}$ by defining a mapping ψ that relies on neighboring timers $t \in \{0,1,2,3\}$ for each edge:

$$\psi(t, t+1) = t \longleftarrow t+1 \pmod 4 \qquad \psi(t+1, t) = t+1 \longrightarrow t \pmod 4$$

There are no arrows if ts differ by 0 or by 2, i.e., the node simply "ignores" such neighbors. We define the local rule function f' to be active on cell i if and only if all (implicit) arrows are pointing to i. More formally, if a cell in state st has an even timer t it mimics $f_{\mathbf{E}}$ and updates t by 2, and if it has an odd timer it mimics $f_{\mathbf{O}}$ and again, updates t by 2 (the extension to Moore's neighborhood is analogous):

$$\begin{aligned} &f'(s_{(1,0)}(t+1), s_{(0,1)}(t+1), st, s_{(-1,0)}(t+1), s_{(0,-1)}(t+1)) = \\ &\qquad f_{\mathbf{E}}(s_{(1,0)}, s_{(0,1)}, s, s_{(-1,0)}, s_{(0,-1)})(t+2) \quad \text{if} \quad t \text{ even} \\ &f'(s_{(1,0)}t_0, s_{(0,1)}t_1, st, s_{(-1,0)}t_2, s_{(0,-1)}t_3) = \\ &\qquad f_{\mathbf{O}}(s_{(1,0)}, s_{(0,1)}, s, s_{(-1,0)}, s_{(0,-1)})(t+2) \quad \text{if} \quad t \text{ odd } \wedge\, t_i + 1 \neq t \;\forall i \end{aligned}$$

Observe that simulated **E** cells will always have an even timer and **O** cells will always have an odd timer. Furthermore, every transition flips all local arrows by increasing the timer, thus ensuring that $\mathcal{A}$ is simulated correctly. Notice that all configurations in $\mathbb{B}$ which correspond to valid encodings of configurations of A embed a FAN, hence will trivially have invariant histories. Let us confirm that our simulation satisfies Definition 15. We can define a bijective unpacking map o with

$$o(c_0)(i,j) = (c_0'(2i,2j), c_0'(2i+1,2j), c_0'(2i,2j+1), c_0'(2i+1,2j+1))$$

which splits a cell of c_0 into 4 different cells in c_0' (one **E** cell and three **O** cells). Since $\mathcal{A}$ simulates A using alternations between **E** and **O** we get $G^A \prec o^{-1} \circ H^B \circ o$. We conclude that $\mathbb{B}$ (asynchronously) simulates A in real time. ◀

We can interpret the mountain-valley encoding as a d-dimensional mountain-valley landscape from Example 10, which slowly travels through space-time, but now, each "mountain" and each "valley" stores the state information too. The results from Proposition 16 can be improved from $4q$ to $3q$ for 1-dimensional automata by introducing a translation inside the simulation. Notice that Moore and von Neumann neighborhood coincide for 1-dimensional automata and are both simply first neighbors $N = (-1, 0, 1)$.

▶ **Proposition 17** (Shifting Mountain-Valleys). *For a synchronous automaton $A = (S, N, f, \mathbb{Z})$ with a first neighbors neighborhood we can construct an asynchronous cellular automaton $\mathbb{B} = (S \times \{1, 2, 3\}, N, f', \mathbb{Z})$ which simulates A in 1.5-linear time.*

Proof. The high-level idea is to improve the construction from Proposition 16 by allowing **E** and **O** nodes to shift in space, which allows us to eliminate one step of our timer (at the cost of slowing down the simulation). Let us consider the mountain-valley landscape as in Example 10, which we visualized as a zigzag line in space-time. Instead of using 2 states, we will now use a tertiary timer to encode "mountains" and "valleys", and disallow nodes with equal values to be neighbors in our initial configuration:

$$(2, 1, 2) \mapsto 3 \qquad (3, 2, 3) \mapsto 1 \qquad (1, 3, 1) \mapsto 2$$

The resulting cellular automaton will still have a mountain-valley structure if launched on periodic configuration 212121212 (see Figure 5). Now let $\mathbb{B} = (S \times \{1, 2, 3\}, N, f', \mathbb{Z})$ be an asynchronous cellular automaton with a timer as described above, and where each state additionally carries simulated state $s \in S$. We encode the transitions of A into $\mathbb{B}$ using the following update function f' ($\oplus/\ominus$ denotes addition and subtraction mod $|S|$):

$$f'(s_{-1}2, s_0 1, s_1 2) = (s_{-1} \oplus s_1)3,$$
$$f'(s_{-1}3, s_0 2, s_1 3) = f(s_{-1} \ominus s_0, s_0, s_1, \ominus s_0)1,$$
$$f'(s_{-1}1, s_0 3, s_1 1) = s_1 2.$$

This execution paradigm can be seen as a mountain-valley landscape where each node has additionally a stage indicator. The stages (coinciding with timer values!) operate as follows:
Stage 1. Compute the $\oplus$ sum of both of your neighbors and move to Stage 3
Stage 2. Compute transition f based on accumulated information and move to Stage 1
Stage 3. Copy state from your right neighbor and move to Stage 2

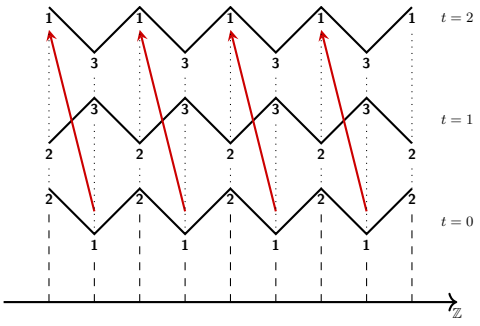

Figure 5 Two synchronous updates of mountain valley landscape represented using a tertiary alternating timer $t \in \{1, 2, 3\}$. Notice that after 3 time steps, the landscape shifts one cell to the left.

Now consider some initial configuration $c_0 = \ldots 0100110 \ldots$ of A which we encode in $\mathbb{B}$ as c'_0 where $c'_0(2i) = c_0(i)2$ and $c'_0(2i + 1) = c_0(i)1$. Observe that after two updates of all cells with odd coordinates and a single update of all cells with even coordinates, we obtain

the encoded configuration $G^A(c_0)$ where the updated states are now placed at positions $2i$ and $2i-1$. Notice, $\mathbb{B}$ cannot have two neighboring transitions active at the same time, hence by Lemma 11, it has invariant histories. By taking the unpacking map o with $o(c_0)(i) = (c'_0(2i), c'_0(2i+1))$ and translation τ with $\tau(c)(i) = c(i-1)$ we observe:

$$(G^A)^2 \prec o^{-1} \circ (H^B)^3 \circ o \circ \tau^2$$

which corresponds to 1.5-linear time simulation. ◀

5 Universality Frontier

Universality is one of the central concepts in computability theory, originally defined in the context of Turing machines [36]. For cellular automata there exist two distinct notions of universality: standard universality, i.e., the ability to "simulate" any Turing machine and *intrinsic universality*, i.e., the ability to "simulate" any cellular automaton (a strictly stronger notion [28]). Since, contrary to Turing machines, cellular automata are both infinite and non-halting by design, a satisfactory definition for universality has always been challenging [29]. We will use the widely accepted definition by Durand [6], which relies on *ultimately periodic configurations* to perform computations (see Appendix of the full version for more details). The exact definition for ultimate periodicity varies across literature [6, 29, 24], hence we present a version general enough to encompass most of them–in particular, the definition below allows to embed d-dimensional configurations into $(d+1)$-dimensional ones, leading to more generalizable universality constructions.

▶ **Definition 18** (Ultimately Periodic Configuration). *A configuration $c \in S^{\mathbb{Z}^d}$ is called ultimately periodic if there exists a $k \in \mathbb{N}$ and a set of scaled standard basis vectors $\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_d \in \mathbb{N}^d$ such that for any j and any cell $i = (i_1, \ldots, i_d) \in \mathbb{Z}^d$ with $i_j > k$ it holds $c(i + \vec{r_j}) = c(i)$.*

Synchronous frontier. Determining the "computation boundary" between universality and non-universality was originally considered by Shannon [34], who initiated the search for the smallest universal Turing machine [26]. These questions also made their way to cellular automata, with the first 2-dimensional universal cellular automaton being constructed by von Neumann [33]. Banks [3] has improved the number of states to just 2 by showing that embedding infinite circuits inside a cellular space $\mathbb{Z}^2$ is sufficient for universality. Later, Minsky showed that Conway's Game of Life cellular automaton is universal [12], and Cook [4] has constructed a universality proof for Rule 110, confirming the existence of a 2-state first neighbors automaton. On the other hand, Wolfram [37] extensively studied elementary cellular automata and concluded that no one-way 2-state universal automata exist (similarly, none exist with just one state or empty neighborhood). He also proposed a candidate 3-state one-way cellular automaton; however, its universality is yet to be proven. The current "universality frontier" for synchronous CAs is given in Figure 1 (left).

Although an explicit definition of asynchronous universality is rarely presented in the literature, all important constructions of universal ACAs essentially assume the slight adjustment of the synchronous definition to the asynchronous scenario [1, 32, 19, 20]. This is consistent with our expectations–by using any simulation technique following Definition 14 on any universal CA, we obtain a universal ACA. Below we present our formalization attempt but, like any other such attempt, it should be taken with a grain of salt.

We call a configuration $c \in S^{\mathbb{Z}^d}$ *computable* if the function $c\colon \mathbb{Z}^d \longrightarrow S$ is computable, and denote with $\lceil c \rceil$ the description of some Turing machine computing c by returning an element of S for every d-dimensional input vector. Similarly, for any description r we denote with $\widetilde{r}$ the underlying configuration. We set $\lceil S^{\mathbb{Z}^d} \rceil$ as the space of all computable descriptions of $S^{\mathbb{Z}^d}$, and observe that it is closed under finite (a)synchronous evolution with any G^ζ.

▶ **Definition 19** (Asynchronous Universality). *An ACA $\mathbb{A} = (S, N, f, \mathbb{Z}^d)$ is called universal if for some partial computable universal function $U\colon \{0,1\}^* \longrightarrow \{0,1\}^* \cup \{\perp\}$ we can find an ultimately periodic configuration c^∞ (compiler) and total computable functions $e\colon \{0,1\}^* \longrightarrow \lceil S^{\mathbb{Z}^d} \rceil$ (encoder) and $d\colon \lceil S^{\mathbb{Z}^d} \rceil \longrightarrow \{0,1\}^*$ (decoder), a pattern $p\colon D \longrightarrow S$ with $D \subset \mathbb{N} \times \mathbb{Z}^d$ finite (halting condition) such that for every input $w \in \{0,1\}^*$:*

1. *The initial configuration $c_0 = \widetilde{e(w)}$ is a finite perturbation of c^∞, i.e. $\|c_0 - c^\infty\| < \infty$ where $\|\cdot\|$ is the l_0 norm on $S^{\mathbb{Z}^d}$.*
2. *If $U(w)$ is defined, then for any update schedule ζ there exists the earliest (asynchronous) time stamp $t \in \mathbb{N}$ such that p appeared in the observed slice of update history h under ζ, and the current configuration c_t^ζ satisfies $d(\lceil c_t^\zeta \rceil) = U(w)$.*
3. *If $U(w) = \perp$ then p will never appear in the update history h under any choice of ζ.*

Asynchronous frontier. Lee [20] kicked off the search for universal asynchronous automata by constructing a 5-state universal von Neumann cellular automaton, which was later improved to 4 states by Lee [19] and to 3 states for Moore's neighborhood by Schneider [32]. By allowing the neighborhood N to be of radius 2 (non-standard case), a 2-state universal automaton was constructed by Adachi [1]. On the other hand, by simulating Rule 110 using improved marching soldiers [38, 21], an 8-state first neighbors universal automaton can be inferred. We improve the frontier by constructing a 3-state von Neumann and a 6-state first neighbors universal automata (presented in Section 7). The updated asynchronous universality frontier is given in Figure 1 (right).

One-way automata. Observe that all simulations mentioned in Section 4 require the neighborhood N to be symmetric, and in fact, no general simulation technique which is applicable to one-way neighborhoods is known. In Section 6 we give an explanation for this by proving that one-way ACAs are, in a sense "equivalent" to finite-state machines. This stands in sharp contrast to the synchronous case, where a simple procedure is known to embed any first neighbors automata into one-way automata (see Appendix of the full version). Thus, we can make the following–somewhat unexpected–observation: the choice of asynchronous setting directly affects the computational capabilities of one-way ACAs. We summarize updated results on the computational power of ACAs under various asynchronous settings in Figure 6.

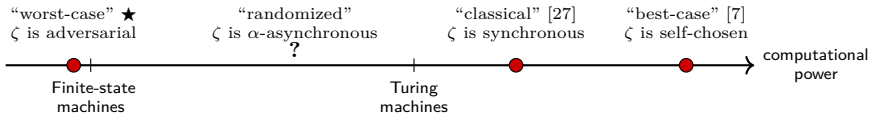

Figure 6 Computational power of one-way ACAs varies with respect to the choice for asynchronous setting. Notice that there are instances of intrinsically universal synchronous one-way CAs, hence they can be seen as more powerful than Turing machines.

▶ **Open question 2.** *What is the computational power of α-asynchronous one-way ACAs?*

6 Non-Universality of One-Way Automata

▶ **Observation 20.** *Asynchronous one-way cellular automata are computationally equivalent to finite-state machines, hence are not universal.*

Since there are many universal synchronous one-way automata (see Figure 1), as an immediate consequence, we get the following result.

▶ **Theorem 21.** *Asynchronous one-way cellular automata are not capable of simulating synchronous one-way cellular automata.*

We prove Theorem 21 and Observation 20 by constructing the required embeddings. Lemma 22 demonstrates that a finite-state machine can replace the evolution of a one-way asynchronous automaton, while Lemma 23 shows the converse by embedding an arbitrary finite-state transducer into an asynchronous one-way cellular automaton (see Appendix of the full version).

▶ **Lemma 22.** *Given a one-way asynchronous cellular automaton $\mathbb{A} = (S, (-1, 0), f, \mathbb{Z})$ we can find an update schedule ζ such that the computation of $\mathbb{A}$ on any input w can be simulated by a semi-automaton $B = (S, S \cup \{\#\}, \delta)$ with S states and alphabet $S \cup \{\#\}$.*

Proof. We assume that we are given an ultimately periodic initial configuration c_0 with the input w encoded into it. Since $\mathbb{A}$ is 1-dimensional we have

$$c_0 = \dots p_L p_L p_L \hat{w} p_R p_R p_R \dots$$

for some p_L and p_R independent of w, and $\hat{w}$, an encoding of w. Now we construct an update schedule ζ and a finite-state machine B based on p_L and p_R and $\mathbb{A}$ such that B simulates $\mathbb{A}$. Let c be a configuration, G (asynchronous) global transition function, $D \subseteq \mathbb{Z}$ be an update set, then we say that the state s is a *pseudo-fixed-point* of cell i under D if s reappears infinitely often in the range of G_D, i.e., $|\{k \in \mathbb{N} \mid G_D^k(c)(i) = s\}| = \infty$.

We claim that for any cell i, configuration c and update set D there exists a pseudo-fixed-point s. Indeed, since $|S|$ is finite but the orbit of i is infinite, at least one element $s \in S$ has to occur infinitely often. Now, consider the scenario where the update set $D = \{i\}$. We can determine the pseudo-fixed-point of i on c by considering the sequence

$$c(i),\ f(c(i-1), c(i)),\ f(c(i-1), f(c(i-1), c(i))),\ \dots$$

It is easy to see that if some state s' appears twice in this sequence, it will appear infinitely often; hence it will be a pseudo-fixed-point. Since this simple pseudo-fixed-point only depends on the states $c(i-1)$ and $c(i)$ we will denote it by $\mathrm{pf}(c(i-1), c(i))$. Using this notion, we define the transition function δ for automaton B and $q, s \in S$ as $\delta(q, s) = \mathrm{pf}(q, s)$.

Let $l + 1$ be the first cell that contains an element of $\hat{w}$ in c_0 and let $r - 1$ be the last cell containing an element from $\hat{w}$ (see Figure 7). Now set x_l to be a pseudo-fixed-point of l on configuration c_0 and update set $D = \{-\infty, \dots, l\}$ and for $j \in \{l + 1, \dots, r - 1\}$ let $x_j = \mathrm{pf}(x_{j-1}, c_0(j))$. Observe that if we choose the initial state x_l for B and execute it on the first $j - l$ letters of $\hat{w}$, then B will be in the state x_j! We construct the update schedule ζ by repeating steps 1–3 shown in Figure 7 indefinitely (every cell is updated infinitely often):

1. Repeatedly update the set $\{-\infty, \dots, l\}$ until cell l is in state x_l
2. Repeatedly update $\{l + 1\}$ until cell $l + 1$ is in state x_{l+1}, then repeatedly update $\{l + 2\}$ until cell $l + 2$ is in state x_{l+2}, and so on until you reach x_r
3. Update the set $\{r, \dots, \infty\}$ once

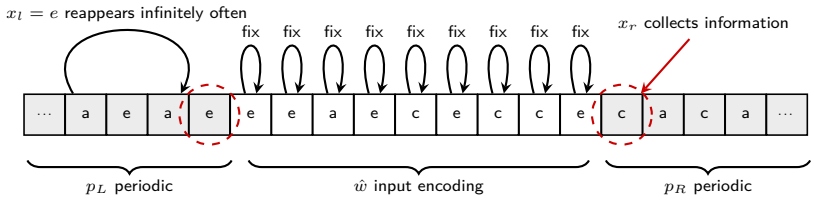

Figure 7 Evolution of $\mathbb{A}$ on c_0 under ζ. Region p_L cycles between the pseudo-fixed-point of x_l, and, so does every cell in $\hat{w}$, hence the cells in p_R only receive constant information on $\hat{w}$.

We make the following observations about the space-time evolution of c_0 under ζ:

- After each iteration of steps 1–3 in ζ the states at positions $l, \ldots, r-1$ do not change and correspond to values $x_l, \ldots, x_{r-1}$
- Evolution of p_L region is independent of $\hat{w}$ and p_R region accesses $\hat{w}$ only through x_{r-1}

Since all states $x_{l+1}, \ldots, x_{r-1}$ are computed by B on $\hat{w}$ and the update history h of cells $\{r, \ldots, \infty\}$ only depends on $x_{r-1} \in S$ and p_R (fixed), the right side of any configuration is only aware of bounded amount of information about $\hat{w}$. Thus, for any possible state of x_{r-1}, we can add a single transition $\delta(x_{r-1}, \#)$ to B, which represents the "outcome" of the joint evolution of p_R and x_{r-1}. We conclude that the ζ evolution of c_0 based on the choice of $\hat{w}$ is at most as complex as the application of B to $\hat{w}\#$, hence it cannot be universal. ◀

▶ **Lemma 23.** *For any finite-state automaton $A = (Q, \Sigma, \delta, q_0, F)$ there exists a one-way ACA $\mathbb{B} = (Q \times (\Sigma \cup \{\varnothing\}) \cup \Sigma, (-1, 0), f, \mathbb{Z})$ simulating A.*

7 Small Universal Asynchronous Automata

There have been two main approaches for designing small universal ACAs: simulating a synchronous CA or designing a universal system using inherently asynchronous primitives.

▶ **Theorem 24** (Six-State Universality). *There exists a 6-state first neighbors universal asynchronous cellular automaton.*

Proof. It is sufficient to apply our improved mountain-valley simulation from Proposition 17 to Rule 110. Since the corresponding invariant history will contain the synchronous space-time of Rule 110, by extension, the resulting automaton is universal. ◀

▶ **Theorem 25** (Three-State Universality). *There exists a 3-state von Neumann universal asynchronous cellular automaton.*

To prove Theorem 25 we use the second approach and design a 3-state universal ACA using infinitely periodic delay-insensitive Boolean circuits. This kind of universality proof is known as *circuit universality* and is commonly used to construct some of the smallest universal cellular automata. Due to the classification of Boolean functions by Post [30], we know that any Boolean function f can be implemented as a feed-forward circuit consisting solely of NAND gates (and the fan-out operation). This result can be further strengthened for *planar* circuits (i.e., circuits without wire crossings).

▶ **Proposition 26** (Planar Circuits [14]). *The subset of planar feed-forward circuits implemented using only* NAND *gates implements all Boolean functions.*

It turns out that the ability to embed arbitrary circuits into 2-dimensional cellular automata is already sufficient for universality and was used to prove universality of several CAs, most notably of Life without Death [17].

▶ **Proposition 27** (Circuit Universality [11] [17]). *A 2-dimensional CA which can implement any Boolean function on a finite subset of cells can embed the space-time of any first neighbors CA, where both use an ultimately periodic initial configuration, hence is universal.*

The proof for Proposition 27 is given in Appendix of the full version, and it naturally generalizes to an asynchronous setting, as long as the ACA allows for reliable asynchronous implementation of Boolean functions. Universality proofs in [20, 19, 1, 32] accomplished this by constructing custom *delay-insensitive* gates, i.e. gates that do not require simultaneity of inputs and operate under arbitrary delays [22]. For asynchronous inputs, a common tactic is to use *dual-rail* encoding, i.e., encode each variable `I` using two "sub-wires" `I` and `¬I` (one for "true" signals, and the other for "false" signals). We take a similar approach and present a simple set of delay-insensitive gates MERGE, FORK, DUAL, and CROSS, as defined in Figure 8.

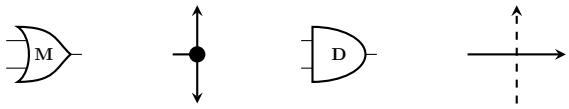

Figure 8 From left to right: MERGE (joins signals from two wires into one, at most one input wire is allowed to carry signal), FORK (splits a signal into two output copies), DUAL (produces an output signal if signals arrive on both inputs) and CROSS (weak wire crossing, simultaneous signals on both wires are prohibited).

▶ **Lemma 28.** *The set of delay-insensitive gates* {MERGE, FORK, DUAL, CROSS} *can compute any Boolean function using a feed-forward planar circuit with dual rail encoding.*

Proof. Due to Proposition 26 it is sufficient to show that we can construct a NAND gate and fan-out operation using a planar circuit. We employ dual-rail encoding and for fan-out FORK both inputs `I` and `¬I` individually and then CROSS them using the fact that at most one of them contains the signal. The circuit for the dual rail NAND is given in Figure 9. ◀

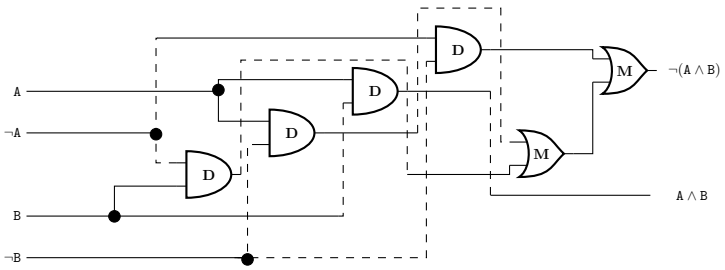

Figure 9 A planar construction for dual rail NAND using 2 MERGE, 4 FORK, 4 DUAL and 12 CROSS gates. Neither MERGE nor CROSS can have two particles arrive at the same time.

As a direct consequence it follows that it is sufficient to find a 3-state ACA $\mathbb{X} = (\{0,1,2\}, N, f, \mathbb{Z}^2)$ with von Neumann neighborhood capable of reliably simulating all delay insensitive gates from Figure 8. We start by describing a "helper" ACA $\mathbb{X}' = (\{0,1,2\}, N, f', \mathbb{Z}^2)$ embedding a FAN and capable of implementing wires, FORK and DUAL. We assume 0 as a

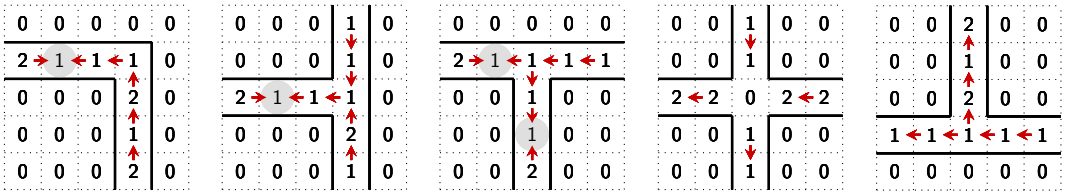

Figure 10 A wire transmitting a signal (highlighted in gray) to the right and to the bottom (first). An implementation of FORK splitting a signal into two copies (second) and an implementation of DUAL asynchronously combining two signals into one (third). A CROSS gate flips 0 to 1 on a signal coming from the left and flips 0 to 2 on a signal coming from the bottom (forth). A MERGE flips the center cell from 1 to 2 if a signal arrives from the top or from the left (fifth).

"background" cell, i.e., a cell that never changes its state and defines no directional indicators. For the remaining nodes we define a mapping ψ similarly to Proposition 16, but now the directional indicator between adjacent non-zero cells points left (or down) if the states are equal, and right (or up) if they are distinct:

$$\psi(a,a) = a \longleftarrow a \qquad\qquad \psi(a,b) = a \longrightarrow b \quad \text{for } a \neq b$$

We use the following principle (similarly to Example 10) to describe the transition function f' of $\mathbb{X}'$: as soon as all directional indicators point to some cell $i \in \mathbb{Z}^2$ it becomes active and can "flip" its state from 1 to 2 or from 2 to 1 respectively. Observe that this state change also causes all adjacent directional indicators to flip. To visualize the underlying circuit, we interpret any active cell of $\mathbb{X}'$ as containing the "signal", and in this way the signal is always transmitted reciprocally to directional indicators, which provides us with a simple way of encoding wires into configurations of $\mathbb{X}'$.

A possible wire construction is shown in Figure 10 (first), and it is not unique–by swapping all 2s with 1s, we get the same wire carrying the same signal, hence there are multiple equally valid ways of transporting signals (this is crucial for implementing crossings). We can easily construct wires into arbitrary directions, for example:

$$1 \longleftarrow 1 \longleftarrow 1 \longleftarrow 1 \longleftarrow 1 \longleftarrow 1 \qquad \text{and} \qquad 2 \longrightarrow 1 \longrightarrow 2 \longrightarrow 1 \longrightarrow 2 \longrightarrow 1$$

describe the left-right wire and the right-left wire, respectively. In a similar fashion, we can implement DUAL by combining two signals into one and FORK by splitting a signal into two, as shown in Figure 10 (second and third). However, ACA $\mathbb{X}'$ can not yet be considered universal because the implementation for MERGE and CROSS seems to be impossible. Hence we augment the transition function f' by adding four additional transitions to obtain the final ACA $\mathbb{X}$ (we use □ for 0, ■ for 1 and ■ for 2).

Notice that the new active transitions do not affect wires, FORK or DUAL from Figure 10. The new MERGE functions like a DUAL, but it is sufficient to flip just one directional indicator to transmit a signal, see Figure 10 (fifth). Similarly, for CROSS, as soon as one signal arrives at the crossing, it gets immediately transmitted by "flipping" the 0 cell and without creating any interference, as shown in Figure 10 (forth). Multiple signals arriving simultaneously can cause potential problems, but this is forbidden by definitions of MERGE and CROSS.

We note that this implementation of gates "breaks" them as soon as they have processed a signal, but this will not have any effect on feed-forward circuits, since those do not contain loops. It is clear that as long as wires have a spacing of at least two 0 cells in between, any

feed-forward circuit embedded in $\mathbb{X}$ operates correctly, hence is universal. We verified our construction using the `Golly` [35] simulator (implementation provided in Appendix of the full version).

▶ Remark 29. Notice that due to introduction of MERGE and CROSS gates, $\mathbb{X}$ no longer has a "dual" flip automata network as in Lemma 11, therefore it does not guarantee invariant histories. Nevertheless, it still has invariant histories on all initial configurations which correspond to valid circuits.

References

1 Susumu Adachi, Jia Lee, and Ferdinand Peper. Universal 2-state asynchronous cellular automaton with inner-independent transitions. *Natural Computing*, pages 107–116, 2010. doi:10.1007/978-4-431-53868-4_12.

2 Ivan Baburin, Matthew Cook, Florian Grötschla, Andreas Plesner, and Roger Wattenhofer. Universality frontier for asynchronous cellular automata, 2025. doi:10.48550/arXiv.2502.05989.

3 Edwin Roger Banks. Universality in cellular automata. *11th Annual Symposium on Switching and Automata Theory (swat 1970)*, pages 194–215, 1970. doi:10.1109/SWAT.1970.27.

4 Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1), 2004. URL: http://www.complex-systems.com/abstracts/v15_i01_a01.html.

5 Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. *Programmability of Chemical Reaction Networks*, pages 543–584. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/978-3-540-88869-7_27.

6 Bruno Durand and Zsuzsanna Róka. The game of life: Universality revisited. *Cellular Automata*, pages 51–74, 1999. doi:10.1007/978-94-015-9153-9_2.

7 Nazim Fatès. A guided tour of asynchronous cellular automata. *Cellular Automata and Discrete Complex Systems*, pages 15–30, 2013. doi:10.1007/978-3-642-40867-0_2.

8 Li-Juan Fei, Jia Lee, Xin Huang, and Ferdinand Peper. Effect of random fluctuations on minimizing the complexity of universal asynchronous cellular automata. *Physica D: Nonlinear Phenomena*, 428:133052, 2021. doi:10.1016/j.physd.2021.133052.

9 Ehud Fonio, Yael Heyman, Lucas Boczkowski, Aviram Gelblum, Adrian Kosowski, Amos Korman, and Ofer Feinerman. A locally-blazed ant trail achieves efficient collective navigation despite limited information. *eLife*, 5:e20185, November 2016. doi:10.7554/eLife.20185.

10 Edward Fredkin and Tommaso Toffoli. Conservative logic. *Collision-based Computing*, pages 47–81, 2002. doi:10.1007/978-1-4471-0129-1_3.

11 Anahí Gajardo and Eric Goles. *Circuit Universality of Two Dimensional Cellular Automata: A Review*, pages 131–151. World Scientific, 2007. doi:10.1142/9789812770837_0007.

12 Martin Gardner. Mathematical games. *Scientific American*, 223(4):120–123, October 1970. doi:10.1038/scientificamerican1070-120.

13 Aviram Gelblum, Ehud Fonio, Yoav Rodeh, Amos Korman, and Ofer Feinerman. Ant collective cognition allows for efficient navigation through disordered environments. *eLife*, 9:e55195, May 2020. doi:10.7554/eLife.55195.

14 Leslie M. Goldschlager. The monotone and planar circuit value problems are log space complete for p. *Acm Sigact News*, 9(2):25–29, 1977. doi:10.1145/1008354.1008356.

15 Eric Goles and Servet Martínez. Automata networks. *Neural and Automata Networks*, pages 15–37, 1990. doi:10.1007/978-94-009-0529-0_2.

16 Ulrich Golze. (a-)synchronous (non-)deterministic cell spaces simulating each other. *Journal of Computer and System Sciences*, 17(2):176–193, 1978. doi:10.1016/0022-0000(78)90003-X.

17 David Griffeath and Cristopher Moore. Life without death is p-complete. *Complex Systems*, 10(6), 1996. URL: http://www.complex-systems.com/abstracts/v10_i06_a03.html.

18 Peter Gács. Deterministic computations whose history is independent of the order of asynchronous updating, 2001. arXiv:cs/0101026.

19 Jia Lee, Susumu Adachi, Ferdinand Peper, and Shinro Mashiko. Delay-insensitive computation in asynchronous cellular automata. *Journal of Computer and System Sciences*, 70(2):201–220, 2005. doi:10.1016/j.jcss.2004.10.009.

20 Jia Lee, Susumu Adachi, Ferdinand Peper, and Kenichi Morita. Embedding universal delay-insensitive circuits in asynchronous cellular spaces. *Fundam. Informaticae*, 58(2003):295–320, 2003. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi58-3-4-07.

21 Jia Lee, Susumu Adachi, Ferdinand Peper, and Kenichi Morita. Asynchronous game of life. *Physica D: Nonlinear Phenomena*, 194(3):369–384, 2004. doi:10.1016/j.physd.2004.03.007.

22 Jia Lee, F. Peper, S. Adachi, and S. Mashiko. Universal delay-insensitive systems with buffering lines. *Ieee Transactions on Circuits and Systems I: Regular Papers*, 52(4):742–754, 2005. doi:10.1109/TCSI.2005.844229.

23 Kristian Lindgren and Mats G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Syst.*, 4(3), 1990. URL: http://www.complex-systems.com/abstracts/v04_i03_a04.html.

24 Kenichi Morita. *Theory of Reversible Computing*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2017. doi:10.1007/978-4-431-56606-9.

25 Katsuo Nakamura. Asynchronous cellular automata and their computational ability. *Systems, computers, controls*, 5(5):58–66, 1974.

26 Turlough Neary and Damien Woods. The complexity of small universal turing machines: A survey. *Sofsem 2012: Theory and Practice of Computer Science*, pages 385–405, 2012. doi:10.1007/978-3-642-27660-6_32.

27 N. Ollinger and G. Richard. Four states are enough! *Theoretical Computer Science*, 412(1):22–32, 2011. Complexity of Simple Programs. doi:10.1016/j.tcs.2010.08.018.

28 Nicolas Ollinger. The intrinsic universality problem of one-dimensional cellular automata. *Stacs 2003*, pages 632–641, 2003. doi:10.1007/3-540-36494-3_55.

29 Nicolas Ollinger. *Universalities in Cellular Automata**, pages 189–229. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-540-92910-9_6.

30 Emil L. Post. *The Two-Valued Iterative Systems of Mathematical Logic*. Princeton University Press, Princeton, 1942. doi:10.1515/9781400882366.

31 Gaétan Richard. Rule 110: universality and catenations. In *Proceedings of the First Symposium on Cellular Automata "Journées Automates Cellulaires"*, Regular paper track, pages 141–160, Uzès, France, April 2008. ISBN 978-5-94057-377-7. URL: https://hal.science/hal-00273995.

32 Oliver Schneider and Thomas Worsch. A 3-state asynchronous ca for the simulation of delay-insensitive circuits. *Cellular Automata*, pages 565–574, 2012. doi:10.1007/978-3-642-33350-7_58.

33 Jacob T. Schwartz, John von Neumann, and Arthur W. Burks. Theory of self-reproducing automata. *Mathematics of Computation*, 21(100):745, 1967. doi:10.2307/2005041.

34 Claude E. Shannon. *A Universal Turing Machine with Two Internal States*, pages 157–166. Princeton University Press, Princeton, 1956. doi:10.1515/9781400882618-007.

35 Andrew Trevorrow and Tom Rokicki. Golly (program). https://golly.sourceforge.io/. Accessed: 2024-11-14.

36 A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. doi:10.1112/plms/s2-42.1.230.

37 Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002. URL: https://www.wolframscience.com.

38 Thomas Worsch. Towards intrinsically universal asynchronous ca. *Natural Computing*, 12(4):539–550, 2013. doi:10.1007/s11047-013-9388-3.