# On the complexity of two-dimensional signed majority cellular automata

Eric Goles, Pedro Montealegre, Kévin Perrot, Guillaume Theyssier

# On the complexity of two-dimensional signed majority cellular automata

Eric Goles[a,b], Pedro Montealegre[c], Kévin Perrot[d], Guillaume Theyssier[e]

[a] *Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile.*
[b] *Le Studium, Loire Valley Institute for Advanced Studies, Orléans, France*
[c] *Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, Orléans, France.*
[d] *Aix-Marseille Université, CNRS, LIF UMR 7279, 13288 Marseille, France.*
[e] *Aix Marseille Université, CNRS, I2M UMR 7373, 13288 Marseille, France.*

**Abstract**

We study the complexity of signed majority cellular automata on the planar grid. We show that, depending on their symmetry and uniformity, they can simulate different types of logical circuitry under different modes. We use this to establish new bounds on their overall complexity, concretely: the uniform asymmetric and the non-uniform symmetric rules are Turing universal and have a P-complete prediction problem; the non-uniform asymmetric rule is intrinsically universal; no symmetric rule can be intrinsically universal. We also show that the uniform asymmetric rules exhibit cycles of super-polynomial length, whereas symmetric ones are known to have bounded cycle length.

*Keywords:* cellular automata dynamics, majority cellular automata, signed two-dimensional lattice, Turing universal, intrinsic universal, computational complexity

## 1. Introduction

In this paper we study the dynamics of signed majority (synchronous) cellular automata in two dimensional regular grids with the von Neumann neighborhood. They are a particular case of threshold automata, *i.e.* each local function being a threshold one. Concretely, each cell carries a $-1$ or a 1 and evolves depending on whether the pondered sum of its neighbors is positive or not.

This class contains the well-known majority vote (all positive weights) and minority vote (all negative weights) cellular automata [1, 2]. We consider both uniform and non-uniform weights and distinguish between symmetric and non-symmetric ones (*i.e.* whether the mutual weights of two neighboring cells are equal).

To our knowledge the first models taking into account majority functions in discrete networks were related with the physical problems of spin glasses and bootstrap percolation. The spin glasses problem appears in statistical mechanics as a model of magnetization. Roughly speaking, two spins of magnetic atoms $S_i$ and $S_j$ are coupled via an interaction $w_{ij}S_iS_j$. The sign of $w_{ij}$ may be $\pm 1$. Spins change trying to minimize the global quantity $-\frac{1}{2}\sum_{i,j=1}^{N} w_{ij}x_ix_j$. The ground state corresponds to the minimum of a quadratic expression over $\{-1,+1\}^n$. Algorithms, and complexity characterization of such discrete problems have been given, e.g., [3, 4]. Among the local strategies to find the minimum, the most usual corresponds to the local majority, *i.e.* the spin takes the most represented orientation in its neighborhood.

The majority operator appears also in the bootstrap percolation problem [5]. Given a grid with a proportion of sites in state 1, such that a site in state 1 remains forever in this state, the bootstrap percolation problem consists in determining the proportion of 1's necessary to contaminate the whole grid with this state when strict majority is applied at every node.

For the majority operator, it was proved that the class of symmetric threshold networks updated in parallel admits only fixed points and cycles of period two [6]. Further, at the heart of this proof, lies an energy operator associated to the automaton (and similar to the spin glasses energy) which drives the parallel dynamics. For the particular case of symmetric signed majority functions, since states belong to the hypercube $\{-1,+1\}^n$ and weights belong to the set $\{-1,0,1\}$, convergence occurs in time $O(n^2)$ where $n$ is the size of the network. Signed majority has also been used to model artificial neural networks. In [7, 8] Shingai studied the dynamical behavior of one and two dimensional grid with von Neumann neighbors and bounded border conditions. He proved that in a bounded one-dimensional threshold automata every possible cycle has period less than four and in the two dimensional case there exists a bounded asymmetric threshold automaton with unbounded cycles.

On the other hand in [1] C. Moore introduced a decision problem for the

majority automaton acting in a d-dimensional grid: given an initial configuration, determine if a specific cell will change its state during the parallel dynamics within some time bound. In this context, he proved that for $d \geq 3$ this problem is **P**-complete by reducing to the monotone circuit problem, *i.e.* with specific configurations of the automaton it is possible to convey information and to built logical gates as well as crossover in order to simulate arbitrary monotone circuits which is a well know **P**-complete problem. The two dimensional case with von Neumann neighborhood remains open. Recently, for bootstrap percolation, it was proved that the associated decision problem is in the class **NC** [9]. Further, it was also proved that the same decision problem for the usual majority is **P**-complete over planar graphs [10].

Our study follows this line of research and establishes new results concerning the complexity of these kind of systems inspired by physics or biology. However, following the trend of natural computing, we also consider them as potential computing devices and aim at classifying them according to the type of computations they can handle. In order to do this, we put a new look on the embedding of Boolean circuits into cellular automata. Since the first constructions of von Neumann in the 60s, encoding of logical circuits into CA has been the main tool to show various kind of complexity hardness or universality results [1, 11, 12]. Usually, a global hardness result is claimed (P-completeness of prediction, Turing universality, etc) but a specific circuit construction is hidden in the proof. We argue that, on the contrary, the focus should be put on the circuit simulation itself. For instance, there is no general implication between **P**-completeness and Turing-universality although a suitable circuit simulation implies both: proofs factorize well at the level of circuit simulation. More importantly, we can identify two *modes* of simulation depending on whether a logical gate gadget can be re-used reliably several times or is corrupted after the first passage of information. In particular, we establish that in the re-usable case there is a generic solution to crossing of information (Theorem 4) and that simulating AND plus OR gates is enough (even in our planar setting) to yield intrinsic universality [13, 14].

But our main contribution is to establish various universality and hardness results within the class of signed majority cellular automata:

- We first show that uniform signed rules can be partitioned in three equivalence classes, called **symmetric**, **antisymmetric** and **asym-**

**metric** uniform rules. This implies that the study of the uniform rules is reduced to the study of three local functions.

- any **asymmetric uniform** rule is Turing-universal and has a **P**-complete prediction problem;
- the **symmetric non-uniform** one is also Turing-universal and has a **P**-complete prediction problem, but is provably not intrinsically universal, like any symmetric rule (uniform or not).
- the **non-uniform** signed majority CA is intrinsically universal;

Besides, we show that asymmetric uniform rules posses super-polynomial cycles, contrary to symmetric rules which have only cycles of length at most two.

The paper is organized as follows: in Section 2, we introduce the necessary formalism around the objects we study and notions of complexity and universality. In Section 3, we develop a generic toolbox to derive complexity results from the existence of circuit simulations in any cellular automata. In Sections 4 and 5, we present our results about uniform and non-uniform rules respectively. We conclude and give some research perspectives in section 6.

## 2. Preliminaries

### *2.1. Signed Majority Cellular Automata*

In this paper we study cellular automata (CA) defined over a *regular two dimensional lattice* and a *von Neumann* neighborhood, *i.e.* for a cell $v$ with coordinates $(i, j) \in \mathbb{Z}^2$, the neighbors of $v$, are $N[v] = \{v, v_n, v_e, v_s, v_w\}$ where $v_n = (i, j + 1)$, $v_e = (i + 1, j)$, $v_s = (i, j - 1)$, $v_w = (i - 1, j)$. We will also consider finite cell spaces with periodic boundary conditions, in that case the cell space will be of size $N = n^2$ and the above neighborhood is taken *modulo* $n$.

A two dimensional regular lattice is called a *signed regular lattice* if there exists a *sign matrix* $W = (w_{uv})_{u,v\in\mathbb{Z}^2}$ where for each pair of neighbors $u, v$, the value $w_{uv} \in \{-1, 1\}$ is called the *sign* of $(u, v)$. A *signed majority automaton* (SMCA) is defined as a cellular automaton $\mathcal{A} = (Q, F_W)$ defined over a signed regular lattice with sign matrix $W$, with state set $Q = \{-1, 1\}$ and where the rule is defined as $F_W : \{-1, 1\}^{\mathbb{Z}^2} \to \{-1, 1\}^{\mathbb{Z}^2}$ where

$$F_W(x) = u \mapsto \begin{cases} 1 & \text{if } \sum_{v\in N[u]} w_{uv}x_v > 0, \\ -1 & \text{otherwise.} \end{cases}$$

Let $v = (i, j)$ be a cell in a signed majority automaton, and let $w_c, w_n, w_e, w_s$ and $w_w$ the signs of the edges $(v, v)$, $(v_n, v)$, $(v_e, v)$, $(v_s, v)$ and $(v_w, v)$ respectively. We say that $W_v = (w_c, w_n, w_e, w_s, w_w)$ is the *sign vector* of cell $v$. A SMCA is called *uniform* (USMCA) if each cell has the same sign vector. If the sign matrix $W = (w_{uv})$ is such that $w_{uv} = w_{vu}$ for each pair of neighbors $u, v$ then the corresponding signed majority CA is called *symmetric*. If the condition $w_{uv} = -w_{vu}$ holds for any pair of neighbors then the CA is called *anti-symmetric*. A signed majority CA which is not symmetric nor anti-symmetric is called *asymmetric*,

USMCA are classical cellular automata over state set $Q = \{-1, 1\}$ and von Neumann neighborhood because the same local transition map is applied in each cell. In order to have a common formalism, we will also see the *non-uniform signed majority CA* (NSMCA) as a classical cellular automaton over state set $Q' = Q^6$ and von Neumann neighborhood where a cell not only stores its own state but also its sign vector: the dynamics consists in applying the signed majority locally according to the sign vector which stays unchanged during the evolution. Finally, we can also consider the symmetric NSMCA through a single classical cellular automaton of state set $Q'$ still with with von Neumann neighborhood with the following modification: each cell $u$ stores a sign vector $(w_{uv})_{v \in N[u]}$, but when applying the signed majority at $u$, we use the sign vector $(w'_{uv} = w_{uv}w_{vu})_{v \in N[u]}$ and hence the dynamics is guaranteed to correspond to a symmetric signed majority.

When considering a finite cell space (or equivalently a periodic initial configuration on the $\mathbb{Z}^2$ lattice), the number of possible configurations is finite and thus the evolution eventually enters a cycle whose length is between 1 (fixed-point) and $q^N$ where $q$ is the size of the alphabet and $N$ the size of the cell space. The following theorem (which is a direct corollary of [6, 15, 16]) illustrates an essential difference between symmetric and asymmetric cases.

**Theorem 1** (Main theorem of [6, 15])**.** *If $F$ is a symmetric signed majority cellular automaton (uniform or not) and $x$ is a periodic configuration of size $n$, then there exists $t = \mathcal{O}(n^2)$, such that either $F^t(x) = F^{t+1}(x)$ (fixed-point) or $F^t(x) = F^{t+2}(x)$ (cycle of length 2).*

### *2.2. Computational Complexity and Intrinsic Universality*

In this paper we aim at analyzing the complexity of the behavior of the considered systems but also their ability to compute or simulate other systems. The two aspects are of course correlated and we will use various tools

and notions to account for their complexity and universality. To measure the computational complexity we will use the standard classes: **NC** (polynomial circuits of polylogarithmic depth), **P** (deterministic polynomial time), **PSPACE** (polynomial space) through the corresponding classical truth table reductions ($\mathbf{AC}_0$, **LOGSPACE**, **PTIME** respectively). Moreover, we will consider the following canonical problems for each system $F$.

- prediction problem PRED for $F$

  **input** a finite partial input configuration $c$, a marked cell $z$ , a time step $t$

  **question** what is the the value of $F^t(c)_z$ if it is completely determined by $c$.

- $f$-cycle problem for $F$ where $f$ is a non-decreasing function such that $1 \leq f(n) \leq 2^{O(n)}$

  **input** an input periodic configuration $c$ of period $n \times n$

  **question** is the length of the temporal cycle reached from $c$ strictly greater than $f(n)$?

It is well-known (see for instance [17]) that for general cellular automata (even of dimension one) the first problem is always in **P** and can be **P**-complete, while the second is **PSPACE** when $f$ is also **PSPACE** and can be **PSPACE**-hard. Interestingly, the prediction problem is in fact **NC** (or **LOGSPACE**) for a large class of CAs [18].

We now quickly review the basic concepts of intrinsic simulation theory for cellular automata (see [19, 14] for an in depth introduction). The general idea is to formalize a very strict notion of simulation (or reduction) between cellular automata that preserves computational complexity of various aspects of the CA and also many of its dynamical properties. In essence, $F$ is simulated by $G$ if some space-time rescaling of $F$ can be embedded as a subsystem in some space-time rescaling of $G$. It therefore relies on two definitions: subsystem embedding and rescaling.

If $F$ and $G$ are CAs, we denote by $F \triangleleft G$ the fact that $F$ is obtained by restriction and projection of the states of $G$, formally: $\exists \pi : Q \subseteq Q_G \to Q_F$ surjective such that

$$\overline{\pi} \circ G = F \circ \overline{\pi}$$

where $\overline{\pi} : Q^{\mathbb{Z}^2} \to Q_F^{\mathbb{Z}^2}$ is the cell-wise application of $\pi$. $\triangleleft$ is our notation for subsystem embedding.

Given a rectangular shape $\vec{m} = (m_1, m_2)$ and any alphabet $Q$ we define the following bijection from $Q^{\mathbb{Z}^2}$ to $\left(Q^{m_1 m_2}\right)^{\mathbb{Z}^2}$:

$$B_{\vec{m}}(x)(z_1, z_2) = \big(x(z_1, z_2), \ldots, x(z_1{+}m_1{-}1, z_2), x(z_1, z_2{+}1), \ldots, (z_1{+}m_1{-}1, z_2{+}m_2{-}1)\big)$$

which just recodes any configurations by blocks of shape $m_1$ by $m_2$. Now if $t$ is a positive integer and $\vec{z} \in \mathbb{Z}^2$, we define the rescaling of $F$ of parameters $\vec{m}$ and $t$ as the CA $F^{<\vec{m},t>} = B_{\vec{m}} \circ F^t \circ B_{\vec{m}}^{-1}$.

We finally say that $G$ *simulates* $F$, denoted $F \leq G$, if there are parameters $\vec{m}$,$t$,$\vec{m'}$ and $t'$ such that $F^{<\vec{m},t>} \triangleleft G^{<\vec{m'},t'>}$. We also say that $G$ *strongly simulates* $F$ if there are parameters $\vec{m}$ and $t$ such that $F \triangleleft G^{<\vec{m},t>}$. Then, a CA $G$ is *intrinsically universal*[1] if for any CA $F$ we have $F \leq G$. It can be shown that an intrinsically universal CA can in fact *strongly simulate* any CA [19].

This definition isn't an attempt to formalize *the* definitive notion of universality. It is rather a pragmatic choice: strict enough to allow us to prove negative results and separation of classes [19, 14], and natural enough to encompass many constructions of the literature [13] and make intrinsic universality actually a common property [20, 21]. This approach was also successfully adopted to study other systems like self-assembly tilings [22, 23, 24].

## 3. CAs vs. Circuits: Complexity and Universality Toolbox

In this section, we develop a formal toolbox to deal with circuit simulation in CA in great generality. In particular we define two modes of simulation that fundamentally differ in the hardness result they imply:

- in the re-usable mode, simulation of monotone gates (AND, OR) is equivalent to intrinsic universality even for a planar lattice, and in particular there is a generic solution to crossing of information;

[1] In the standard definition, the possibility to compose $F$ or $G$ with translations is added to the definition of rescaling: we remove this aspect from this paper to simplify and thus get a less general notion of intrinsic universality. However, everything can also be done up to translation everywhere.

- in the weaker mode (no re-usability), intrinsic universality is generally not guaranteed and crossing is essential to have **P**-completeness of prediction and Turing-universality.

This section considers 2D CAs in all generality. In the remainder of the paper we will show that, within the natural class of signed majority automata, we have examples for both modes, including ones that are provably not capable of re-usable simulation while able to simulate any circuitery in the weaker mode. This shows the interest of focusing on the simulation mode. Our framework is formalized in two steps: first, we consider the intermediate object of *dynamical circuits* which adds to classical Boolean circuits a dynamical aspect; second, we define how a dynamical circuit can be concretely embedded in a CA in a very generic way without any supposition on how information is encoded.

### *3.1. Dynamical circuit simulations*

A dynamical circuit is a particular kind of Boolean network with a DAG structure and marked input/output nodes.

**Definition 2.** A *dynamical circuit* is a directed acyclic graph where each node has in-degree at most 2. The *input nodes* are the nodes with in-degree 0. The *output nodes* are the nodes with out-degree 0. Moreover, each node $v$ is labeled by some function $f_v : \{0,1\}^{d_i(v)} \to \{0,1\}$, where $d_i(v)$ denotes the in-degree of $v$. For each input node $v$, $f_v$ is the constant function equal to 0. The *set of gates* of a dynamical planar graph is the set of functions $f_v$. At each time step $t$, each node $v$ of the graph contains some Boolean information $b_v(t) \in \{0,1\}$. The (synchronous) dynamics of the graph defines the value of node $v$ at time $t+1$ by $b_v(t+1) = f_v\big(b_{v_1}(t), b_{v_2}(t)\big)$ where $v_1$ and $v_2$ are the incoming neighbors of $v$, and similarly for vertex with in-degree different from 2. If $p$ is the number of input nodes and $q$ the number of output nodes, the dynamical circuit computes the function $f : \{0,1\}^p \to \{0,1\}^q$, defined by

$$f\big(x_1, \dots, x_p\big) = \big(b_{v_1}(T), \dots, b_{v_q}(T)\big)$$

where $T$ is the depth of the DAG (size of the longest directed path) and values on nodes are initialized by:

$$b_v(0) = \begin{cases} x_i & \text{if } v \text{ is the } i\text{th input node,} \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we say that a dynamical circuit is *layer-synchronous* if nodes of level $i+1$ only have incoming neighbors of level $i$, where the level is defined by the DAG structure of the graph starting at level 0 for input nodes.

*Remark* 3. When the dynamical circuit is "layer-synchronous" then the computed function is exactly the same as the one computed by the classical Boolean circuit corresponding to the same DAG: indeed, the computation progresses level by level until it reaches the output gates synchronously. Besides, the condition on the input nodes functions ensures that the input is given as an "impulse": at time zero input gates receive the input bits, at time one they are reset to zero.

The interest of the definition above relies in the following which is to be compared to the well-known result [25] in the classical circuit complexity setting.

**Theorem 4.** *There is a planar dynamical circuit that realizes the crossing function*

$$\chi(x, y) = (y, x)$$

*using only monotone (AND,OR) gates, and such that inputs and outputs are on the outer face of the graph arranged in the order $X_{IN}$, $Y_{IN}$, $X_{OUT}$, $Y_{OUT}$. This implies that the functions computed by monotone dynamical circuits can be computed by monotone planar dynamical circuits. Moreover the dynamical planar monotone circuit value problem is **P**-complete.*

*Proof.* First, by chaining either AND or OR gates, we can make "delayed" wires that transmit the information but in a given number of steps. Second, using again the wire toolkit, we can make new variants of the OR and AND gates were the inputs come to any pair of sides.

Then we can realize the following kind of planar dynamical graphs with arbitrarily chosen length of wires, depicted in Figure 1. Two segments with different length on the drawings can be transformed into wires with different delays): If a signal comes at $X_{IN}$ it will duplicate, arrive at $A$ and $B$ at the same time, then at $C$ and $D$ at the same time and then the AND gate at the bottom will be triggered and send a signal to $X_{OUT}$.

The same is true for transmission from $Y_{IN}$ to $Y_{OUT}$. By monotonicity of all gates of the gadget the $X_{IN} \to X_{OUT}$ transmission is not affected by the presence of any signal from a potential $Y_{IN} \to Y_{OUT}$ transmission. And the same is true for $Y$ with respect to $X$.

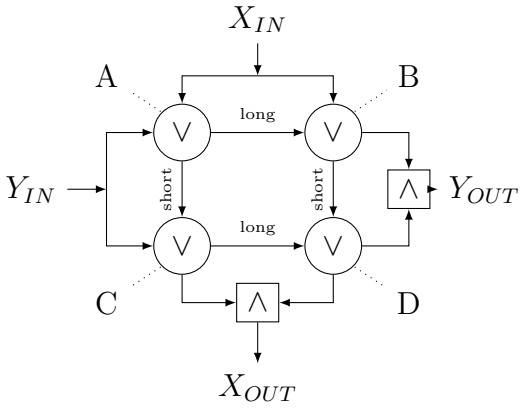



Figure 1: Crossing gadget for planar dynamical circuits with monotone gates.

So, to complete the proof that this gadget realizes crossing, it is sufficient to check that a $X_{IN} \to X_{OUT}$ transmission never triggers the AND gate which outputs to $Y_{OUT}$ (symmetrically for $Y$ with respect to $X_{OUT}$). This AND gate is triggered if and only if at some time both gates $B$ and $D$ output an electron. By choosing horizontal/vertical lengths in the gadget such that $AB = CD = 2BD = 2AC$, the evolution can be characterized by the following successive configurations concerning presence of electrons in different parts of the ABCD rectangle:

| | A | A $\to$ B | A $\to$ C | B | B $\to$ D | C | C $\to$ D | D |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 5 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 6 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

We can check that the simultaneous presence of an electron in B and D never occurs, and deduce that when a signal comes only from $X_{IN}$ it never triggers $Y_{OUT}$.

From the existence of a crossing gadget we deduce P-completeness of the

dynamical planar monotone circuit value problem using the classical encoding used for P-completeness of monotone circuit value problem: each Boolean value $b$ is represented by a positive copy $b$ and a negative copy $\neg b$; then negation is realized by swapping $b$ and $\neg b$ (wire crossing) and the AND/OR gates are realized together with NAND/NOR using De Morgan's laws. □

We now describe two modes of simulation of a set of gates by a cellular automaton. In the first one, the gates can be used only once and hence it can only simulate layer-synchronized circuits. The second one asks for each gate to be re-usable and hence is able to simulate any dynamical circuit. Both modes require the simulation to work in constant time.

The basic simulation mechanism works by using square blocks concatenated in a grid-like fashion that represent parts of the circuit (either a node or an edge). The definition doesn't require any specific way of building this blocks or representing information shared inside. They communicate information with their four neighbors (north, east, south, west) in such a way that each one implements a Boolean function with at most 2 inputs and at most 2 outputs. In the sequel any building block we consider will compute one of the following functions (we represent them using type $\{0,1\}^4 \to \{0,1\}^4$ in order to make explicit the position of inputs and outputs among the neighbors in the order north, east, south, west):

$$\begin{aligned}
\mathrm{AND}(x,*,*,y) &= \big(0,\min(x,y),0,0\big)\\
\mathrm{OR}(x,*,*,y) &= \big(0,\max(x,y),0,0\big)\\
\mathrm{CROSS}(x,*,*,y) &= \big(0,y,x,0\big)\\
\mathrm{NOP}(*,*,*,*) &= \big(0,0,0,0\big)\\
\mathrm{MULTIPLY}(*,*,*,x) &= \big(0,x,x,0\big)\\
\mathrm{WIRE}_{i,o}\big(c\in\{0,1\}^4\big) = k\in\{0,\dots,3\} &\mapsto \begin{cases} c(i) & \text{if } k=o\\ 0 & \text{else}\end{cases}
\end{aligned}$$

for any $i \neq o \in \{0,\dots,3\}$. Note that any function $f$ above is such that $f(0,0,0,0) = (0,0,0,0)$. We denote by $Img(f)$ the set of 4-uple that can be obtained as an image of $f$. The $\mathrm{WIRE}_{i,o}$ functions are just all the possible ways to read a bit on one side and transmit it to another side. Together with the NOP and MULTIPLY function they represent the basic planar wiring toolkit denoted $W$ below. The NOP gate is special in that one considers it has 4 inputs and 4 outputs.

In this paper, in order to simplify statements, we only discuss gates which are not part of the wiring toolkit $W$ because we always assume that gates from $W$ are available in any simulation considered. The following definition of simulation of a gate set $\mathcal{G}$ by a CA precisely requires the considered CA to actually simulate all gates from the set $\mathcal{G} \cup W$.

**Definition 5** (Simulating a gate set)**.** Let $\mathcal{G} \subseteq \{\text{AND,OR,CROSS}\}$ be a set of gates. Let $F$ be a CA with states set $Q$ and $N > 0$ be an integer. Consider a set $V \subseteq Q^{N \times N}$ of patterns, the valid blocks, each of which as a type $f_u$ where $f \in \mathcal{G} \cup W$ and $u \in Img(f)$ and such that, for any $f_u$, there is some block of $V$ of type $f_u$. If a block $B \in V$ has type $f_{(a,b,c,d)}$ for some $f$, we say it has *north value* $a$, *east value* $b$, *south value* $c$ and *west value* $d$. Finally let $\Delta > 0$ be some constant. A configuration is *valid* if it is a concatenation of valid blocks where output sides of a block must face input sides of its corresponding neighbors. Given a block $B \in V$ of type $f_u$ in a valid configuration, we say that it *makes the correct transition* if it becomes a block of type $f_v$ after $\Delta$ steps where $v = f(n, e, s, w)$ is the output of $f$ on the input read from surrounding blocks, precisely: the block at the north of $B$ has south value $n$, the block at the east of $B$ has west value $e$, etc.

**Weak simulation.** We say that $F$ *simulates the set of gates* $\mathcal{G}$ with delay $\Delta$ and valid blocks $V$ if for any valid configuration $c$, the configuration $F^{\Delta}(c)$ is valid and for any $f \in \mathcal{G} \cup W$, any block of type $f_{(0,0,0,0)}$ in $c$ makes the correct transition.

**Re-usable simulation.** The simulation is *re-usable* if any block in any valid configuration makes the correct transition.

*Remark* 6.

- Each block of $V$ has a type $f_{(n,s,e,w)}$ but the definition does not specify the way information is encoded inside blocks: for instance, the 'north value' $n$ is not necessarily determined by the value of a particular cell in the block, it could be encoded as the parity of the number of occurrences of some state in the whole block, or as the $i^{\text{th}}$ binary digit of $\pi$ where $i$ is a sum of some states in the whole block, etc;

- The fact that several valid blocks can have the same type allows some redundancy in the information encoding; the necessity of this redundancy in block encodings is an open problem in the context of intrinsic universality (open problem 6 of [14]); in our context it can be understood intuitively as follows: in general, several contexts can force a

block to evolve to the same type $f_{(n,s,e,w)}$ after $\Delta$ steps, but the information flow and the computation happening inside the block can leave traces of this context inside the block;

- The positions of inputs and outputs around a block to simulate a gate is not strict for OR, AND and MULTIPLY gates since we can rewire without crossing to any choice of positions thanks to the symmetry of this gate; for the CROSS gate, it is sufficient to respect the fact that inputs and outputs are not interleaved.

In the above definition, blocks are used to simulate gates with at most 2 inputs and at most 2 outputs. We naturally want to concatenate these blocks in complicated ways to compute more general Boolean functions with many inputs and outputs. The following lemma says that doing this we are able to realize any function computable by dynamical circuits corresponding to the set of gates and the simulation mode. Said differently, the grid layout and synchronization by wire lengths can be solved in a generic way.

We first formalize how we derive the simulation of functions computed by circuits from the simulation of their gate set. The intuition is that by concatenation of valid blocks we can build meta-blocks that have the desired input/output relation.

**Definition 7** (Simulating a Boolean function)**.** Let $F$ be any CA that simulates a set of gates $\mathcal{G}$ with delay $\Delta$ and valid blocks $V$ and consider some finite function $\phi : \{0,1\}^p \to \{0,1\}^k$. We say that $F$ *simulates* $\phi$ if there are integers $m \geq \max(p,k)$ and $T$, and a collection $\mathcal{V}$ of $m \times m$ valid patterns of blocks from $V$ indexed by outputs of $\phi$

$$\mathcal{V} = \bigcup_{v \in \mathrm{Img}(\phi)} \mathcal{B}_v$$

where $\mathcal{B}_v \subseteq V^{m \times m}$, and such that

1. in each block $B \in \mathcal{B}_v$, the $k$ $V$-blocks starting from the upper-right corner and going downwards must produce the sequence of bits $v$ when reading their east value;

2. a block $B \in \mathcal{B}_{(0,\dots,0)}$, when in a valid configuration, becomes $B' \in \mathcal{B}_{\phi(u)}$ after $T$ steps where $u$ is the input of $p$ bits read in the initial configuration on the left side of $B$ starting from the top.

The simulation is re-usable if the second condition above is also true for any block from $\mathcal{V}$.

**Lemma 8** (Grid layout and synchronization lemma)**.** *Let $F$ be any CA that simulates a set of gates $\mathcal{G}$ with delay $\Delta$ and valid blocks $V$ (Definition 5). We have:*

- *if $\{AND, OR\} \subseteq \mathcal{G}$ then $F$ can simulate (Definition 7) any function computed by a planar monotone layer-synchronous dynamical circuit with inputs and outputs placed on the outer face in a non-interleaved way (inputs on one half, outputs on the other half);*
- *if $\{AND, OR, CROSS\} \subseteq \mathcal{G}$ then $F$ can simulate (Definition 7) any function computed by a monotone layer-synchronous dynamical circuit;*
- *if $\{AND, OR\} \subseteq \mathcal{G}$ and the simulation is re-usable then $F$ can simulate in a re-usable way (Definition 7) any function computed by a monotone dynamical circuit.*

*Proof.* We proceed by a sequence of transformations of a given circuit that preserve the computed function. It is important to note that we don't need any bound on the size of the final simulation. Moreover we have, by definition 5, the existence of wires that allows to draw arbitrary discrete paths on the plane. These two aspects make the proof rather straightforward, without any technical difficulty.

Given a dynamical circuit $C$ with the corresponding hypothesis in each case, we proceed like this:

- first, in the case of re-usable simulation, we can suppose that $C$ is planar by theorem 4.
- we then add as many OR gates with 1 input and out-degree 2 to ensure out-degree 2 for each gate of $C$. While doing this we pad when necessary by useless OR gates (1 input, 1 output) so that there is a constant $k$ such that each arc in $C$ is encoded as a directed path of length $k$ in the new circuit $C'$. In particular $C'$ is layer synchronous if $C$ is, and both compute the same function;
- then, we consider a grid layout of the graph of $C'$: nodes are positioned at points of $\mathbb{Z}^2$ and arcs are polygonal lines. Note that in any case, we

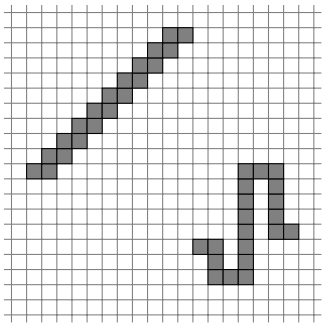


Figure 2: Compensating for the difference between Euclidian lengths by making zig-zags: the two paths have the same $\mathbb{Z}^2$-length but the Euclidian distances between starting and ending positions are clearly different.

position inputs and outputs on the outer face without interleaving. By multiplication of the grid step (embedding $\mathbb{Z}^2$ into $(k\mathbb{Z})^2$), we can choose the minimal distance between two point of interest (node or crossing between lines) to be as large as we need without changing the ratio between length of arcs. In particular, we can ensure that the minimal distance $D$ between two points of interest is much larger than the ratio $\alpha$ between the longest encoded arc and the shortest encoded arc; Moreover, we can ensure that each node is on the grid $(2\mathbb{Z})^2$ so that the $\|\cdot\|_1$-distance between any two nodes is even;

- then, we transform each Euclidean arc by a $\mathbb{Z}^2$-path and at each crossing (if any) we put a crossing gate. To do this while ensuring that all Euclidean arcs are represented by $\mathbb{Z}^2$-paths of the exact same $\mathbb{Z}^2$-length (number of steps on the $\mathbb{Z}^2$ grid), we use 'fast' (almost) minimal $\mathbb{Z}^2$-paths for long Euclidean arcs, and 'slow' zig-zag $\mathbb{Z}^2$-paths for short Euclidean arcs (see Figure 2). The remarks above ensure that we can do it because:

  1. the worst case we have to deal with is a discrete path of length $\alpha$ times the Euclidean length of its corresponding arc: choosing $D >> \alpha$ ensure that we can make zig-zags of sufficient width without collisions;
  2. the fine-tuning of distances is possible because everything is even and we can replace a path of length 2 by a path of length 4 with same in and out points. Moreover, the Euclidean arcs were chosen

long enough so that we can modify their corresponding zig-zag locally on a negligible fraction of their length to implement this fine tuning.

After this process we get a partial labeling of $\mathbb{Z}^2$ with gates from the set $\mathcal{G} \cup W$. To show formally that the function computed by the circuit $C$ is simulated by $F$ it remains to complete a big square containing this partial labeling with NOP gates and rewire inputs and outputs so that they are horizontally aligned. The rewiring is possible even without crossing gate since inputs and outputs are on the outer face without interleaving. □

In the next section, we give an example of a threshold-like CA that can realize such simulations. Before this, we derive various consequences of the existence of each kind of simulation.

*3.2. Three flavors of circuit simulation, three levels of complexity*

We are now going to deduce various hardness results using the existence of some circuit simulation. First, the simple fact of simulating wires in the weakest mode already almost gives $\mathbf{NC}^1$-hardness if one thinks about grid graph reachability problems. The difficulty is that wires alone do not provide any means of keeping trace of some passage of information. We therefore add a FUSIBLE gate, aimed at linking the presence of a signal on a wire at any time, to the prediction problem PRED of a cell's state at some precise time. Indeed, in the particular case of the following $\mathbf{AC}^0$ reduction, the time window during which the signal we wish to predict may arrive is unbounded, though the reduction allows only constant time computation.

$\text{FUSIBLE}_i(x \in \{0,1\}^4) = (0,0,0,0)$ is a gate that contains a particular cell $z$ such that

$$F^t(c)_z = \begin{cases} 1 & \text{for all } t \geq t_0 + \Delta \\ -1 & \text{for all } t < t_0 \\ \pm 1 & \text{otherwise} \end{cases}, \text{ where } t_0 \text{ is the first time at which } x(i) = 1.$$

**Proposition 9.** *If a 2D CA $F$ can simulate $\text{WIRE}_{i,o}$, NOP and $\text{FUSIBLE}_i$ for all $i, o$ then its prediction problem* PRED *is $\boldsymbol{NC}^1$-hard under $\boldsymbol{AC}^0$ reductions.*

*Proof.* The in-degree-one-out-degree-one grid graph reachability problem (11GGR) has been proven to be $\mathbf{NC}^1$-hard under $\mathbf{AC}^0$ reductions in [26]. We prove

the proposition by constructing an $\mathbf{AC}^0$ reduction of 11GGR to our prediction problem. Since the composition of two $\mathbf{AC}^0$ algorithms is an $\mathbf{AC}^0$ algorithm, the result will hold.

Let us first formally define 11GGR. A directed grid graph is a directed graph whose vertices are a subset of $\mathbb{N} \times \mathbb{N}$, and all edges are of the form $(i, j) \to (i + b, j)$ or $(i, j) \to (i, j + b)$ with $b \in \{-1, 1\}$. The 11GGR problem is the following. Given a directed grid graph $\mathbb{G}$ of size $n \times m$ with in-degree and out-degree at most 1, and two vertices $s$ and $t$ of $\mathbb{G}$, does there exists a directed path from $s$ to $t$ in $\mathbb{G}$?

Given such a directed grid graph, the idea of the reduction is straightforward: every directed path is encoded by an analogue wire using $\text{WIRE}_{i,o}$ gates, $s$ is encoded by a 1 impulse, and $t$ is encoded by a FUSIBLE gate, all this on a background of NOP gates. More precisely, the reduction maps every vertex to a gate as follows (see Figure 3 for an illustration):

- $s$ is encoded by a valid block with value 1 on every side;
- $t$ is encoded by a $\text{FUSIBLE}_i$ gate with $i$ the side of the arc whose head is $t$;
- any other vertex $v$ having in-degree and out-degree 1 is encoded by a $\text{WIRE}_{i,o}$ gate, where $i$ is the side of the arc whose head is $v$, and $o$ is the side of the arc whose tail is $v$;
- any other point of the $n \times m$ rectangle is encoded by a NOP gate;

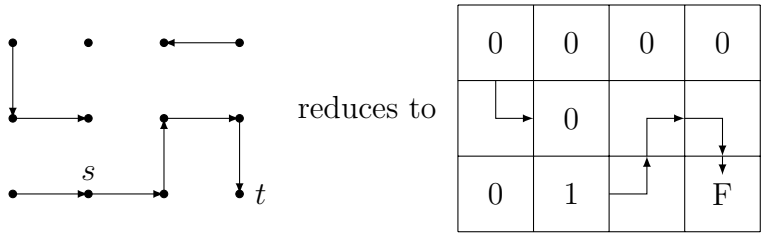



Figure 3: Reduction from 11GGR to the prediction problem on a dynamical circuit with $\text{WIRE}_{i,o}$, NOP and $\text{FUSIBLE}_i$ gates. Wires are indicated by arrows, 1 is a valid block with value 1 on every side (corresponding to $s$), F is the FUSIBLE gate whose input is pointed, and 0 are NOP gates.

The instance of Pred asks for the state of the distinguished cell of the unique FUSIBLE gate at time $(nm + 1)\Delta$, which corresponds to the length

of the longest possible path plus one additional delay for the fusible to melt. This reduction can be computed in parallel (each gate separately) on a CREW-PRAM, using a constant time and $nm$ processors. It can therefore be implemented by a family of circuits in the $\mathbf{AC}^0$ class (polynomial size and bounded depth circuits), according to the time-processors/depth-size correspondance presented in [27]. □

When doing circuit simulations in the weak mode one cannot expect information processing to happen in a fixed region of space since gadget simulating gates are potentially destroyed after their first use. However, it is perfectly possible to handle unbounded one-dimensional computations by having a one-dimensional computation front that progresses in the 2D lattice during the evolution and therefore fires each gadget at a given position only once. For instance, the computation front can be diagonal so that the value of cell $z \in \mathbb{Z}$ at time $t$ of the 1D computing process is read in the block $(t+z, t-z)$ at some time $O(t)$ in the 2D simulating CA. With AND, OR and crossings gates, it is possible to organize arbitrary computations in that way.

**Proposition 10.** *If a 2D CA $F$ can simulate a $\{AND, OR, CROSS\}$ circuitry then it can simulate any 1D CA with a constant time slowdown, and therefore it is Turing-universal and its prediction problem* PRED *is* ***P****-complete.*

*Proof.* Consider any finite set $Q$ and any local function $f : Q^2 \to Q$. There is some $k$ and some encoding $\phi : Q \to \{0,1\}^k$ such that $\phi$ is injective and $Img(\phi)$ does not contain two comparable strings of bits for the componentwise order $\prec$ (e.g. $10 \prec 11$ but $01 \not\prec 10$). Therefore there is a monotone function $f' : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k$ that simulates $f$ up to the recoding $\phi$, *i.e.*

- $u_1 \prec v_1$ and $u_2 \prec v_2 \Rightarrow f'(u_1, u_2) \prec f'(v_1, v_2)$;
- $\forall q_1, q_2 \in Q, \phi(f(q_1, q_2)) = f'\big(\phi(q_1), \phi(q_2)\big)$.

Then there is a monotone layer-synchronous dynamical circuit that computes $f'$ (because there is a classical circuit that does) so, by Lemma 8, $f'$ is computed by $F$ using valid blocks of the circuit simulation. Moreover, using wires and crossings and the constructions of Lemma 8, we can construct a valid meta-block that receives inputs $x$ and $y$ on its west and south sides (respectively) and produces $f'(x, y)$ both on its north and east sides (see

figure 4). Using zig-zag wiring techniques as in lemma 8 we can ensure that all logical connections are realized by paths of the same length so that both outputs of the meta block are produced at the same time when inputs come in synchronously. Since $f'$ simulates $f$ through the encoding $\phi$, we can associate

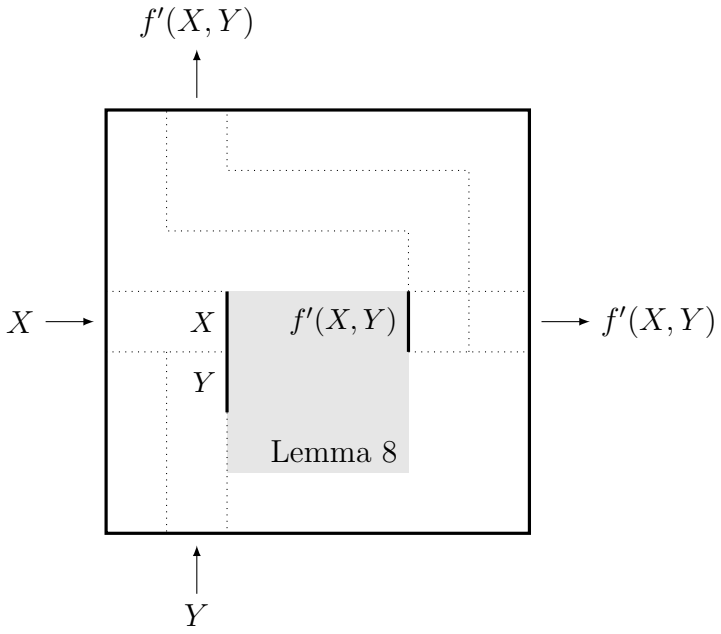



Figure 4: A block simulating the local transition function of a 1D CA.

a set of meta blocks $B_q$ to each $q \in Q$. Then, starting from any configuration $C : \mathbb{Z}^2 \to Q$ representing a concatenation of aligned meta blocks on a meta $\mathbb{Z}^2$ grid (meta block of type $B_{C(z)}$ at position $z \in \mathbb{Z}^2$), the CA $F$ simulates on each diagonal the 1D CA with two neighbors and local transition function $f$. Formally, it can be expressed at the meta block level by

$$F^T(C)_{(i,j)} \in B_{f(C_{(i-1,j)},C_{(i,j-1)})}$$

where $T$ is the time constant common to all meta blocks.

Choosing $f$ and $Q$ appropriately we deduce Turing-universality of $F$ (it is straightforward to build a 1D CA with 2 neighbors that simulates a given Turing machine).

Choosing $f$ and $Q$ so that the corresponding 1D CA has a $P$-complete prediction problem we deduce $P$-completeness of the prediction problem of

$F$. For instance, we can take the 2-block recoding of elementary CA 110 which is P-complete [28], *i.e.* $Q = \{0,1\}^2$ and $f$ such that:

$$f\big((x_0,x_1),(x_2,x_3)\big) = \big(f_{110}(x_0,x_1,x_2), f_{110}(x_1,x_2,x_3)\big).$$

The reduction of the prediction problem of this 1D CA to the prediction problem of $F$ is a truth table reduction done as follow: to an input $(w,t,z)$ for the prediction problem in the 1D CA made of a 1D word $w \in Q^*$ of size $n$ a time $t$ and a position $z \in \mathbb{Z}$ we associate the finite set of input $(W, Tt, z_j)_j$ where

- $W$ is a $n \times n$ pattern of meta blocks which are all the same $B_{q_0}$ except on the diagonal where $B_{w_i}$ is at position $(i,-i)$;
- $z_j = (K(z+Tt), -Kz) + \epsilon_j$ where $K$ is the size of the meta blocks and $\{\epsilon_j\} = [0,K-1] \times [0,K-1]$ are all the finite positional offsets needed to visit all cells of the meta block.

The simulation by diagonals ensures that the reduction is correct: knowing the entire meta-block at meta-position $(z+Tt,-z)$ at time $Tt$ allows to know its type and therefore the value of cell $z$ after $t$ iteration of the 1D CA on input $w$. This reduction is also clearly logspace computable (the size and complexity of meta blocks is just a constant in the reduction). □

The next proposition formalizes what has been present informally in the literature since a long time[12, 13]: for 2D CA, intrinsic universality is equivalent to the ability to simulate a rich enough re-usable circuitry. Using Theorem 4, we can state it in a stronger way where only monotone gates are required.

**Proposition 11.** *A 2D CA $F$ can simulate in a re-usable way the gates $\{AND, OR\}$ if and only if it is intrinsically universal. In that case its prediction problem is **P**-complete and its $f$-cycle problem is **PSPACE**-hard for any $f$ with $1 \leq f(n) \leq 2^{O(n)}$.*

*Proof.* First, if $F$ is intrinsically universal then it is strongly intrinsically universal [14], so in particular it can strongly simulate any CA which is well known for its ability to simulate AND OR gates in a re-usable way, for instance the CA of Banks [12] which has 2 states and von Neumann neighborhood, or the well-known "Game of Life" [29]. The ability to simulate

a set of gates is preserved by strong simulation (the simulating CA does if the simulated CA does).

For the other direction, consider some $F$ that simulates gates $\{\text{AND}, \text{OR}\}$ in a re-usable way. We proceed in a similar way as in proof of Proposition 10, except that all simulations here are re-usable. Consider any finite set $Q$ and any $\phi : Q^4 \to Q$. It can be encoded into a monotone function $\phi' : \left(\{0,1\}^k\right)^4 \to \{0,1\}^k$ which can be computed by a dynamical monotone circuit because it can be computed by a classical monotone circuit. Then, using lemma 8, we know that $F$ can simulate in a re-usable way $\phi'$ using valid blocks. From that construction we can obtain, by rewiring and crossing (lemma 8 actually implies that a re-usable crossing meta block is constructible), a meta block that receive one input from each side encoding a state of $Q$, computes $\phi'$ on this 4-uple and sends the result to each side (see Figure 5). Since $\phi'$ simulates $f$ through the encoding $\phi$, we can associate a

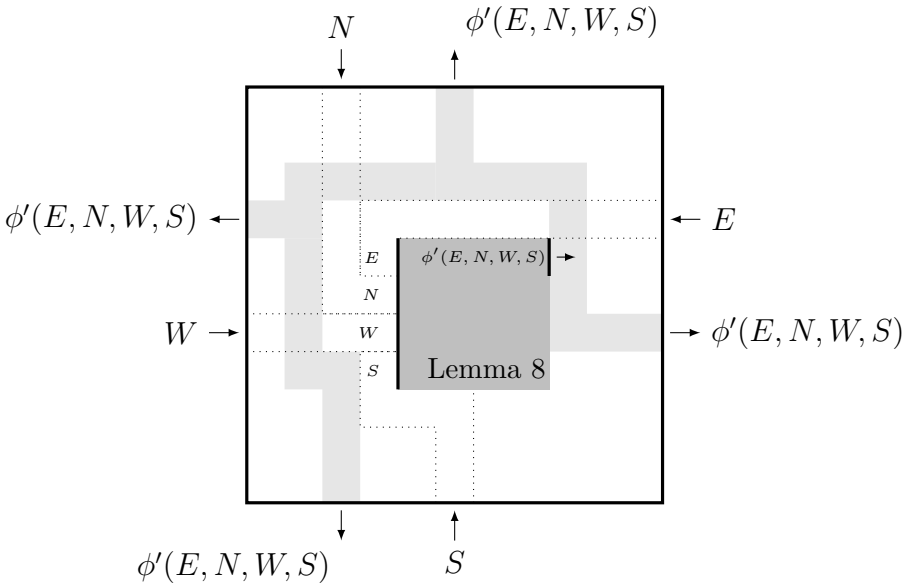



Figure 5: A block simulating in a re-usable way the local transition function of a 2D CA.

set of meta blocks $B_q$ to each $q \in Q$. Then, starting from any configuration $C : \mathbb{Z}^2 \to Q$ representing a concatenation of aligned meta blocks on a meta $\mathbb{Z}^2$ grid (meta block of type $B_{C(z)}$ at position $z \in \mathbb{Z}^2$), the CA $F$ simulates the CA with von Neumann neighborhood and local transition function $f$.

Formally, it can be expressed at the meta block level by

$$F^T(C)_{(i,j)} \in B_{f(C_{(i+1,j)},C_{(i,j+1)},C_{(i-1,j)},C_{(i,j-1)})}$$

where $T$ is the time constant common to all meta blocks.

Choosing $f$ to be intrinsically universal, we deduce that $F$ is intrinsically universal. Now consider a 2D CA $F'$ with the following properties:

- its states set allows to form structured $N \times N$ patterns, where some $O(N)$ part encodes an instance of the quantified Boolean formula problem (QBF) (see for instance [30]), some $O(N)$ part is reserved to all the necessary computation to test the existence of a solution to the problem, and a $\Omega(N \times N)$ part allows to encode a binary counter of length $\Omega(N \times N)$;
- starting from a valid pattern as above, it starts to search (by brute force) for a solution to the QBF problem;
- if it doesn't find any solution, it erases everything with a spreading state, so that it reaches a fixed point;
- if it finds a solution, it start to increment the binary counter at each step forever (the counter restart to 0 when its maximal value is reached), so that it enters a temporal cycle of length $\Omega(2^{N \times N})$.

It is straightforward to design such a CA $F'$ with a simple enough structure so that the $f$-cycle problem of $F'$ is **PSPACE**-complete for any $f$ with $1 \leq f(n) \leq 2^{O(n)}$. Indeed, the CA $F'$ converges to a cycle of length 1 (hence no larger than $f(n)$), or a cycle of length $\Omega(2^{N \times N})$ (hence strictly larger than $f(n)$ for large enough $n$) depending on whether the QBF instance has a solution or not. The fact that QBF is **PSPACE**-hard (see for instance [30]) allows us to conclude. □

## 4. Uniform Signed Majority Cellular Automata

In this section we will study USMCA both in their dynamical properties and their complexity. Recall that USMCA are signed majority cellular automata where each cell has the same sign vector.

There are 32 possible sign vectors, hence 32 different USMCA. However, from those 32 USMCA only 12 are different up to rotation. We subdivide these

| Name | Sign vector $(w_c, w_n, w_e, w_s, w_w)$ | Group |
|---|---|---|
| $F_1$ | $(+1,+1,+1,+1,+1)$ | Symmetric |
| $\overline{F}_1$ | $(-1,-1,-1,-1,-1)$ | Symmetric |
| $F_2$ | $(+1,-1,-1,-1,-1)$ | Symmetric |
| $\overline{F}_2$ | $(-1,+1,+1,+1,+1)$ | Symmetric |
| $F_3$ | $(+1,-1,+1,-1,+1)$ | Symmetric |
| $\overline{F}_3$ | $(-1,+1,-1,+1,-1)$ | Symmetric |
| $F_4$ | $(+1,+1,-1,-1,+1)$ | Antisymmetric |
| $\overline{F}_4$ | $(-1,-1,+1,+1,-1)$ | Antisymmetric |
| $F_5$ | $(+1,+1,-1,+1,+1)$ | Asymmetric |
| $\overline{F}_5$ | $(-1,-1,+1,-1,-1)$ | Asymmetric |
| $F_6$ | $(+1,-1,-1,-1,+1)$ | Asymmetric |
| $\overline{F}_6$ | $(-1,+1,+1,+1,-1)$ | Asymmetric |

Table 1: List of the six Symmetric, two Antisymmetric, and four Asymmetric USMCAwith their respective sign vectors.

12 USMCA into the following three groups, according to the symmetries of the sign matrix $W = (w_{uv})_{u,v\in\mathbb{Z}^2}$. We say that a USMCA is *symmetric* if $W$ is a symmetric matrix, i.e. $w_{uv} = w_{vu}$ for each pair of cells $u, v$. On the other hand, we say that a USMCA is *antisymmetric* if $W$ satisfies that $w_{uv} = -w_{vu}$ for each pair of cells $u, v$. When the USMCA is neither symmetric nor antisymmetric, we say that it is *asymmetric*. There are six Symmetric, two Antisymmetric, and four Asymmetric USMCA, shown in Figures 6, 7 and 8, respectively.

Some rules are equivalent under simple transformations of configurations and these transformations preserve their ability to simulate gates sets. One could imagine a lot of simulation-preserving transformations, but the following will be sufficient for our purpose. A *block permutation* is a bijective transformation of $Q^{\mathbb{Z}^2}$ which consists in partitioning the space into $n \times n$ blocks in a $\mathbb{Z}^2$-regular way, applying a common permutation $\pi$ of $Q^{n\times n}$ on each $n \times n$ block of the configuration. Such a transformation is given by $n$, $\pi$ and the alignment of blocks. Two CAs $F$ and $F'$ are *block conjugate* if there is a block permutation $\Phi$ such that $F \circ \Phi = \Phi \circ F'$. The following lemma shows that block-conjugacy preserves simulation of gate sets. The informal intuition of this fact is very simple: if $F$ is block-conjugate to $F'$ via $\Phi$ and

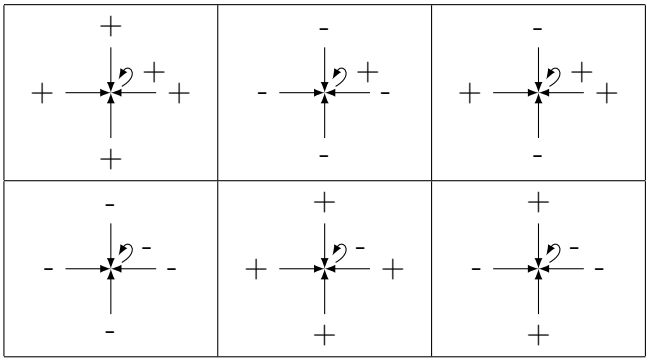


Figure 6: The six Symmetric USMCA are called, from left to right in the first line $F_1, F_2, F_3$ and $\overline{F}_1, \overline{F}_2, \overline{F}_3$ in the second line.

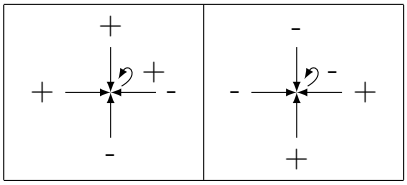


Figure 7: The Antisymmetric USMCA are called from left to right: $F_4$ and $\overline{F}_4$.

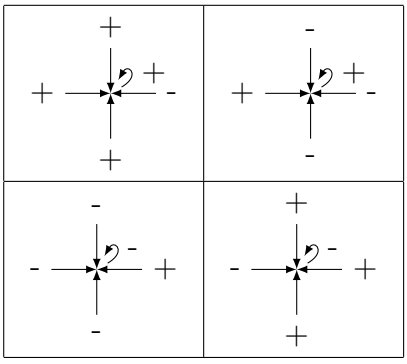
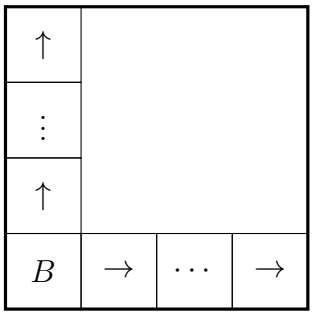


Figure 8: The four Asymmetric USMCA are called $F_5, F_6$ (in the top line) and $\overline{F}_5, \overline{F}_6$ (in the bottom line).

can simulate a gate set using a set of valid blocks $V$, then $F'$ simulates the same gate set using the blocks obtained by applying $\Phi$ to the blocks of $V$. However there is a technical complication: if the size of blocks of $V$ and the size of blocks used by $\Phi$ are multiplicatively independent then several alignments occur. This is solved by expanding blocks of $V$ to a convenient size which is always possible thanks to the freedom allowed in Definition 5.

**Lemma 12.** *If two CAs $F$ and $F'$ are block-conjugate then they can simulate the same sets of gates under the same mode.*

*Proof.* Let $\Phi$ be the block permutation giving the conjugacy $F' \circ \Phi = \Phi \circ F$ and denote by $n_0$ the size of blocks it uses. Suppose that $F$ can simulate a set of gates with valid blocks $V_F$ of size $N \times N$ and delay $\Delta$. For each block $B \in V_F$ (supposing that $B$ has outputs at north and east, other cases are similar) we define the corresponding block $B'$ of size $n_0N \times n_0N$ as follows:




where arrows correspond to wires (with input/output positions indicated by the arrows) and the blank part is filled with NOP blocks. We get a set $V'_F$ of blocks and it is straightforward to check that $F'$ simulates the same set of gates with blocks $V'_F$ and delay $n_0\Delta$ (the type given to $B'$ is the type of $B$). Now let $B''$ be the block obtained by applying $\Phi$ to $B'$ with the correct alignment (the lower-left corner of $B'$ coincides with the lower-left corner of a block of $\Phi$). We get a set $V_F$ of blocks of size $n_0N \times n_0N$ with the following property: any concatenation of blocks of $V_F$ is the image under $\Phi$ of the corresponding concatenation of blocks of $V'_F$ and, after $n_0\Delta$ steps of $F$ and $F'$ respectively, this correspondence is preserved between images. We deduce that $F$ simulates the set of gates under the same mode. The lemma follows because block-conjugacy is a symmetric relation. □

The following lemma shows that there are only 3 essentially different rules up to block-conjugacy.

**Lemma 13.** *Let $F$ be a symmetric (resp. antisymmetric, resp. asymmetric)* USMCA *then $F$ is block-conjugate to $F_1$ (resp. $F_4$, resp. $F_5$).*

*Proof.* Let $\alpha : \{-1,1\}^{\mathbb{Z}^2} \to \{-1,1\}^{\mathbb{Z}^2}$ be defined as $\alpha(x) = -x$. We begin showing that for every $i \in \{1,2,3,4,5,6\}$, any configuration $x \in \{-1,1\}^{\mathbb{Z}^2}$ and any $t \geq 0$, we have that $F_i^t = \alpha^t \circ \overline{F}_i^t$. Indeed, notice that if $W$ is the sign matrix of rule $F_i$ and $\overline{W}$ is the sign matrix of rule $\overline{F}_i$, then $\overline{W} = -W$. This implies that for every $v \in \mathbb{Z}^2$

$$\sum_{u \in N(v)} w_{uv} x_u = -\sum_{u \in N(v)} (-w_{uv}) x_u \Rightarrow F_i(x) = -\overline{F}_i(x)$$

and inductively,

$$\begin{aligned} \textstyle\sum_{u \in N(v)} w_{uv} (F_i^t(x))_u &= \textstyle\sum_{u \in N(v)} w_{uv} \cdot \left[ (-1)^t \cdot (\overline{F}_i^t(x))_u \right] \\ &= (-1)^{t+1} \textstyle\sum_{u \in N(v)} (-w_{uv}) \cdot (\overline{F}_i^t(x))_u \\ &\Rightarrow F_i^{t+1}(x) = (-1)^{t+1} \overline{F}_i^{t+1}(x) = \alpha^{t+1}(\overline{F}_i^{t+1}(x)). \end{aligned}$$

Consider now the function $\beta : \{-1,1\}^{\mathbb{Z}^2} \to \{-1,1\}^{\mathbb{Z}^2}$ such that

$$\beta(x)_{(i,j)} = \begin{cases} -x_{(i,j)} & \text{if } i+j \text{ is even,} \\ x_{(i,j)} & \text{if } i+j \text{ is odd.} \end{cases}$$

Then $F_1^t = \beta \circ F_2^t \circ \beta$ and $\overline{F}_1^t = \beta \circ \overline{F}_2^t \circ \beta$. Indeed, let $W = (w_{uv})_{u,v \in \mathbb{Z}^2}$ be the sign matrix of $F_1$ (respectively $\overline{F}_1$) and $W' = (w'_{uv})_{u,v \in \mathbb{Z}^2}$ be the sign matrix of $F_2$ (respectively $\overline{F}_2$). Notice that for every $u \neq v$ we have that $w_{uv} = -w'_{uv}$. Then for every $v = (i,j) \in \mathbb{Z}^2$ if we write

$$\sum_{u \in N(v)} w_{uv} x_u = \begin{cases} -\left[ \displaystyle\sum_{u \in N(v), u \neq v} (-w_{uv}) \cdot x_u + w_{vv}(-x_v) \right] & \text{if } i+j \text{ is even,} \\ \displaystyle\sum_{u \in N(v), u \neq v} (-w_{uv}) \cdot (-x_u) + w_{vv} x_v & \text{if } i+j \text{ is odd.} \end{cases}$$

we obtain $F_1(x) = \beta((F_2(\beta(x)))$ and $\overline{F}_1(x) = \beta((\overline{F}_2(\beta(x)))$. Since $\beta \circ \beta = id$, then for any $t \geq 0$, $F_1^t = \beta \circ F_2^t \circ \beta$ and $\overline{F}_1^t = \beta \circ \overline{F}_2^t \circ \beta$.

Consider finally the function $\gamma : \{-1,1\}^{\mathbb{Z}^2} \to \{-1,1\}^{\mathbb{Z}^2}$ such that

$$\gamma(x)_{(i,j)} = \begin{cases} -x_{(i,j)} & \text{if } i \text{ is even} \\ x_{(i,j)} & \text{if } i \text{ is odd} \end{cases}$$

Then $F_1^t = \gamma \circ F_3^t \circ \gamma$, $\overline{F}_1^t = \gamma \circ \overline{F}_3^t \circ \gamma$, $F_5^t = \gamma \circ F_6^t \circ \gamma$ and $\overline{F}_5^t = \gamma \circ \overline{F}_6^t \circ \gamma$.

Indeed, let $F = F_1$ (resp. $\overline{F}_1$, $F_5$, $\overline{F}_5$) and $F' = F_3$ (resp. $\overline{F}_3$, $F_6$, $\overline{F}_6$), let $W = (w_{uv})_{u,v\in\mathbb{Z}^2}$ be the sign matrix of $F$ and $W' = (w'_{uv})_{u,v\in\mathbb{Z}^2}$ the sign matrix of $F'$ . Clearly for any $(i,j)$, we have that $w_c = w'_c$, $w_e = w'_e$ and $w_w = w'_w$, while $w_n = -w'_n$ and $w_s = -w'_s$. If we write for $v = (i,j)$,

$$\sum_{u\in N(v)} w_{uv}x_u = \begin{cases} -[w_c(-x_v) + (-w_n)x_{v_n} + w_e(-x_{v_e}) \\ \qquad +(-w_s)x_{v_s} + w_w(-x_{v_w})] & \text{if } i \text{ is even,} \\ w_c x_v + (-w_n)(-x_{v_n}) + w_e x_{v_e} \\ \qquad +(-w_s)(-x_{v_s}) + w_w x_{v_w} & \text{if } i \text{ is odd.} \end{cases}$$

Then clearly $F = \gamma \circ F' \circ \gamma$, and since $\gamma \circ \gamma = id$, we conclude that $F^t = \gamma \circ (F')^t \circ \gamma$ □

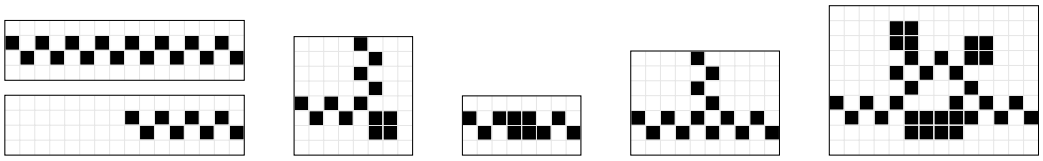
**Theorem 14.** *Any symmetric rule can simulate planar monotone layer-synchronous dynamical circuit, and has a $\boldsymbol{NC}^1$-hard prediction problem. However, it cannot simulate re-usable monotone circuitry and is not intrinsically universal.*

*Proof.* Figure 9 presents how $F_1$ can simulate planar monotone layer synchronous dynamical circuit (Lemma 8): $W$, AND and OR gates. Thanks to the additional FUSIBLE gate, Proposition 9 allows to conclude the $\mathbf{NC}^1$ hardness of $F_1$, hence of all symmetric rules (Lemma 13). Concerning the second part of the statement, Theorem 1 implies that the $f$-cycle problem is trivial for symmetric rules when $f$ is constant equal to 3. By proposition 11 these rules cannot be intrinsically universal and cannot simulate re-usable AND OR circuitry. □

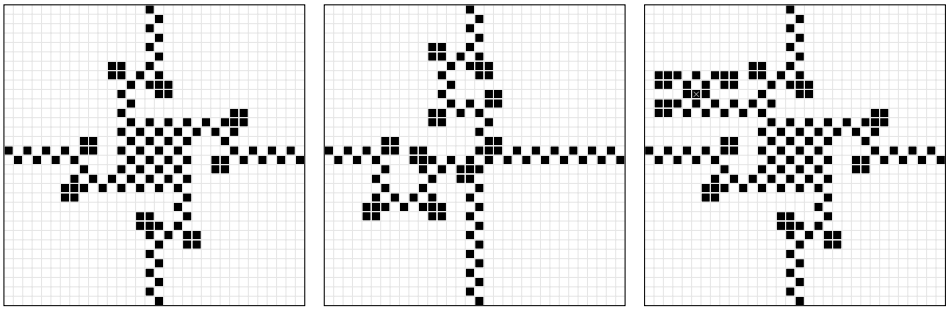
The symmetric rules are limited due to their trivial behavior on limit cycles. The following theorems show that antisymmetric and asymmetric rules are not as trivial concerning the limit cycles.

**Theorem 15.** *Any antisymmetric signed majority cellular automaton can exhibit limit cycles of length $\Omega(N)$, where $N$ is the number of sites.*

*Proof.* From Lemma 13 we know that it is enough to prove the result for rule $F_4$. Let $n \geq 16$ be such that $(n-12)/4$ is even, and consider the following configuration in a grid of size $n \times n$: in the first line of the grid repeat $(n-12)/4$ times the pattern $(1,1,-1,-1)$. Then, in the 12 remaining cells, repeat twice the pattern $(1,1,-1,-1,-1,-1)$. Then, in the consecutive



(a) A wire is constantly blinking (left top) and transmits a signal by being consumed (left bottom). A turn (center left). An obstructor prevents a signal (center). A multiplier (center right). These gadgets can straightforwardly be composed to simulate a wiring toolkit $W$. Plus a diode (right).



(b) OR (left), AND (center) and FUSIBLE (right, with a distinguished cell) gates. These gates have a particular feature: the four sides are undifferentiated inputs/outputs. For example, the AND gates waits for any two signals to arrive, and then sends a signal to the two remaining sides. One can easily transform them into classical gates using diodes.

Figure 9: $F_1$ simulates planar monotone layer-synchronous dynamical circuit. Animations are presented on Appendix A.

lines, copy the pattern of the previous line and shift the pattern one cell to the west. Finally, pick a special line $i$, and in the two locations of the line $i$ with four consecutive sites in state $-1$, replace $(-1,-1,-1,-1)$ by $(-1,-1,1,-1)$, call this line a *perturbation*. In Figure 10 there is an example for $n = 20$. This configuration satisfies that the perturbation is shifted two sites to the north and two sites to the west every $2n$ steps. Therefore, a limit cycle of length $N = n^2$ is obtained. □

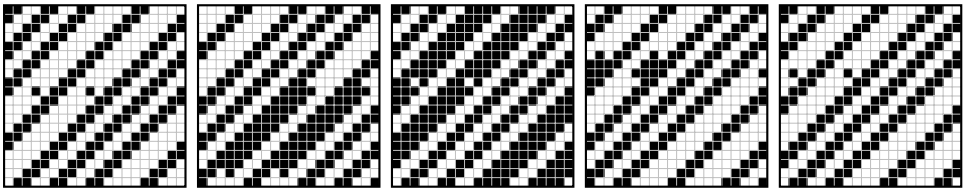


Figure 10: Example of a pattern of size $n \times n$ for $n = 20$, that exhibits a limit cycle of length $n^2$ when updated according to the antisymmetric rule $F_4$. In the figure are represented steps $0, 10, 20, 30$ and 40, from left to right. Note that at step 40 the configuration has shifted two sites to the north and two sites to the west.

**Theorem 16.** *Any asymmetric signed majority cellular automaton can exhibit limit cycles of super-polynomial length.*

*Proof.* From Lemma 13, it is enough to show long limit cycles just for rule $F_5$. We start showing how to build a *cycle gadget with k perturbations*, which in a finite periodic configuration of size $8 \times n$ exhibits limit cycles of length $n$ if $k = 0$ and limit cycles of length $n(n/2 + k)$ if $k > 0$. Figure 11 shows the cycle gadget with zero perturbation: a particle moves through a *wire*: a straight horizontal line of cells initially in state $-1$. In the top of this line, there is a series of straight lines of three cells alternating states. In the bottom, another series of straight lines alternating in inverse order. In the middle line two consecutive cells in state 1 form a particle, which in each step moves one cell to the east.

A *perturbation* corresponds to change the state from 1 to $-1$ to a cell in the top line of the wire, and $k$ *perturbations* correspond to change the same changes to $k$ consecutive cells in the top line of the wire (see Figure 12). Each perturbation delays the particle by two time steps compared to the case without perturbation. Moreover, each time that the particle passes through a perturbation, the configuration shifts two cells to the west. For a cycle gadget with $k > 0$ perturbations, we obtain limit cycles of length $n/2(n + 2k) = n(n/2 + k)$.

We can compose several cycles gadgets with different numbers of perturbations, as shown on Figure 13. Notice that the dynamics of one cycle gadget do not affect the others. The length of the composition's limit cycle is then the least common multiple of the limit cycles of each cycle gadget.

Let $\pi(n)$ be the number of prime numbers in $(n/2, n)$. Consider the composition of every cycle gadget with perturbation in $\{p_1 - n/2 \ldots, p_{\pi(m)} - n/2\}$, such that $\{p_1 \ldots, p_{\pi(m)}\}$ are all the prime numbers in $(n/2, n)$. We obtain a periodic configuration of size $N = n \times m$, where $m = \mathcal{O}(\frac{n}{\log n})$, and with a limit cycle of length $n \cdot \mathrm{lcm}(p_1, \ldots, p_{\pi(m)}) = 2^{\Omega(n)} = 2^{\Omega(N^{1/3})}$ (which can be deduced from the Prime Number Theorem [31]). □

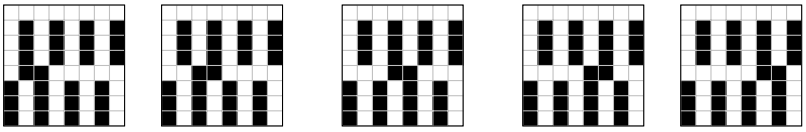


Figure 11: Example of 5 steps of cycle gadget with zero perturbations, for $n = 8$. The state 1 (resp. $-1$) is represented as a black cell (resp. white cell). The moving particle takes $n$ steps to return to the initial configuration.

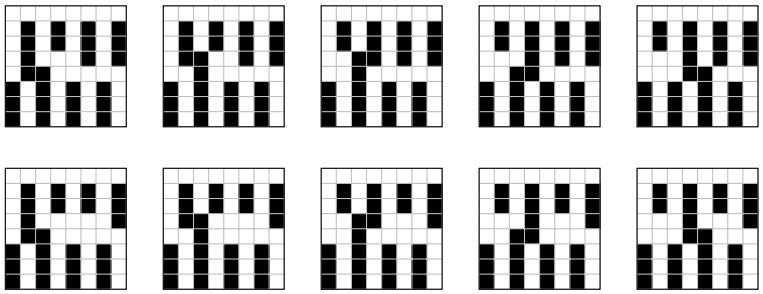


Figure 12: Examples of perturbations in the cycle gadgets for rule $F_5$. The first line represents the dynamics of a cycle gadget of size $n = 8$ with one perturbation, notice that the limit cycle reached in this case has period $n(n/2+1)$. On the second line is represented the first 5 steps in the dynamics of the cycle gadget with two perturbations, reaching a cycle of length $n(n/2 + 2)$.

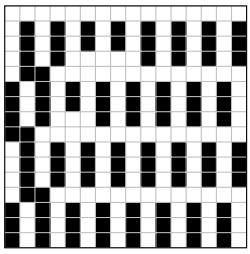


Figure 13: Composition of three cycle gadgets with zero, one, and two perturbations. The length of the limit cycle in this case is $n \cdot \mathrm{lcm}(n/2 + 1, n/2 + 2)$

**Theorem 17.** *Any asymmetric signed majority CA can (weakly) simulate* $\{AND, OR, CROSS\}$ *circuitry. It is therefore Turing-universal and has a* $\boldsymbol{P}$*-complete prediction problem.*

*Proof.* From Lemma 13 we know that it is enough to prove the result for rule $F_5$. In Figures 16 and 17 we present the elements required to apply Proposition 10. Given the intrinsic asymmetry of this rule, different gadgets are required to transmit information in the different orientations. In Figure 16 are presented the 12 possible wires, while in Figure 17 we present the gate gadgets. Remark that we do not present explicitly the CROSS gadget, but enough tools in order to build it. Indeed, we present the following functions:

$$(\neg west)\textsc{And}(north)(x, *, *, y) = (0, (1 - y)x, 0, 0),$$

$$(west)\textsc{And}(\neg north)(x, *, *, y) = (0, (1 - x)y, 0, 0),$$

from which we can compute a XOR gate gadget using the formula.

$$(P)\textsc{xor}(Q) = (P \wedge \neg(P \wedge Q)) \vee (Q \wedge \neg(P \wedge Q)).$$

We simulate the CROSS gate using the gadgets on Figures 14 and 15. □

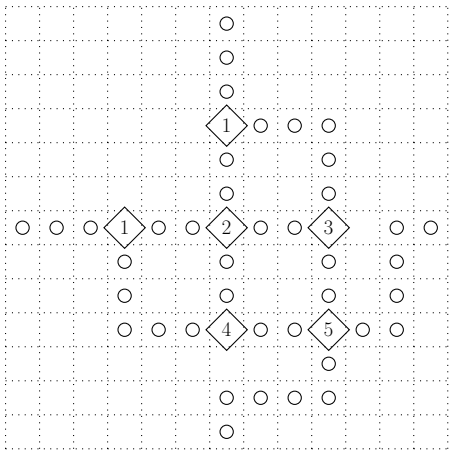



Figure 14: Representation of a XOR gate. Diamonds numbered 1 represent a MULTIPLY gate, the diamond numbered 2 represent a AND gate (with two outputs using a MULTIPLY gadget). The diamonds numbered 3 and 4 represent $(\neg west)\textsc{And}(north)$ and $(west)\textsc{And}(\neg north)$ gates, respectively (Figure 17(b)). The diamond numbered 5 represent an OR gate (with two outputs using a MULTIPLY gadget). Finally, the circular shaped nodes are wires (Figure 16).

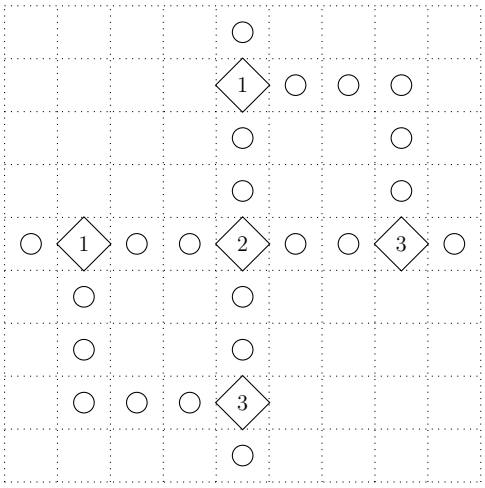
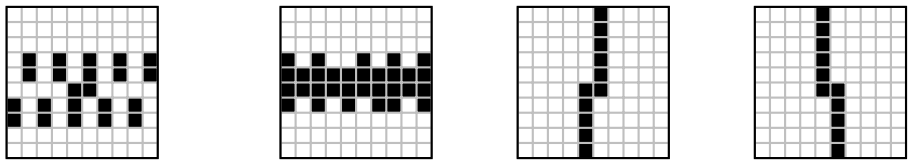
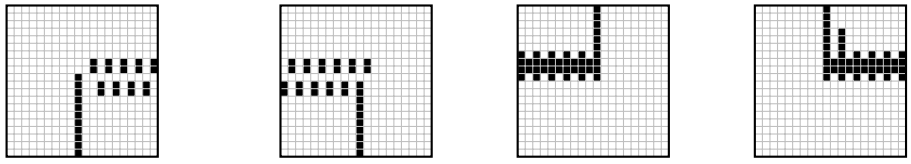


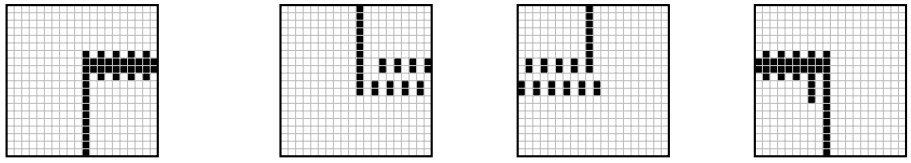

Figure 15: CROSS gate. Diamonds 1 represent MUTLIPLY gates. The diamonds numbered 2 and 3 represent a XOR gadgets.



(a) $\text{WIRE}_{w,e}$ (b) $\text{WIRE}_{e,w}$ (c) $\text{WIRE}_{n,s}$ (d) $\text{WIRE}_{s,n}$

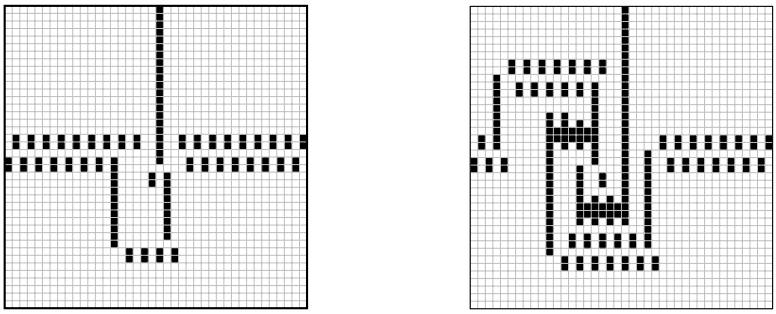


(e) $\text{WIRE}_{s,e}$ (f) $\text{WIRE}_{w,s}$ (g) $\text{WIRE}_{n,w}$ (h) $\text{WIRE}_{e,n}$

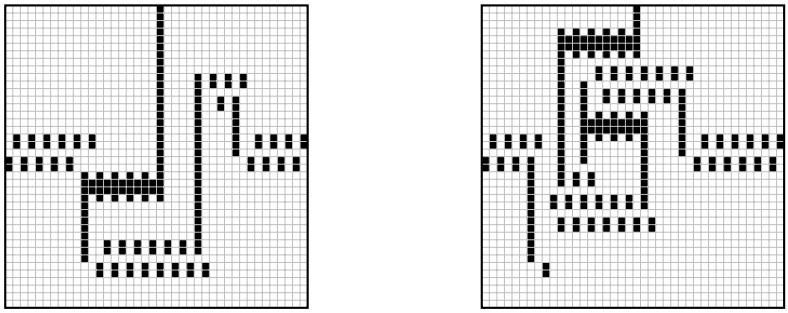


(i) $\text{WIRE}_{e,s}$ (j) $\text{WIRE}_{n,e}$ (k) $\text{WIRE}_{w,n}$ (l) $\text{WIRE}_{s,w}$

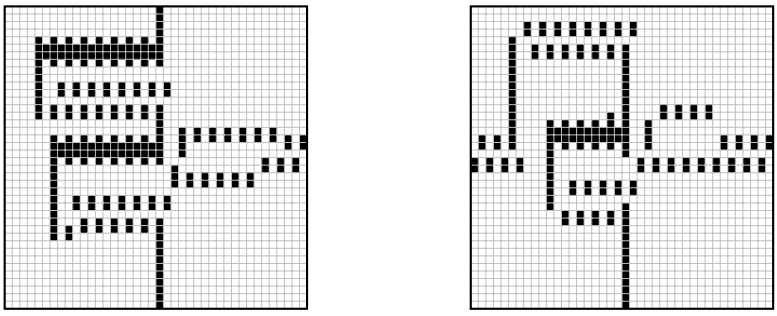
Figure 16: Wires for the rule $F_5$ (asymmetric). Recall that $\text{WIRE}_{x,y}$ is a wire from direction $x$ to direction $y$ with $x, y \in \{n \text{ (north)}, e \text{ (east)}, s \text{ (south)}, w \text{ (south)}\}$. The state 1 (resp. $-1$) is represented as a black (resp. white) cell. Subfigures 16a to 16d represent the block during its evolution after a bit 1 arrived to the corresponding side. The other subfigures show the block in its initial state before any information arrives. Animations are presented on Appendix B.



(a) AND (left) and OR (right) gates. The inputs are from north and west, and the outputs at east.



(b) $(\neg west)\textsc{And}(north)$ and $(west)\textsc{And}(\neg north)$ gates. These gates receive inputs from north and west, and output to the east.



(c) MULTIPLY gadgets, with inputs from the north and west sides.

Figure 17: Gates for the rule $F_5$. The state 1 (resp. $-1$) is represented as a black (resp. white) cell. CROSS gadget can be constructed using the gadgets presented on Figure 17b. Animations are presented on Appendix C.

## 5. Non-Uniform Signed Majority Cellular Automata

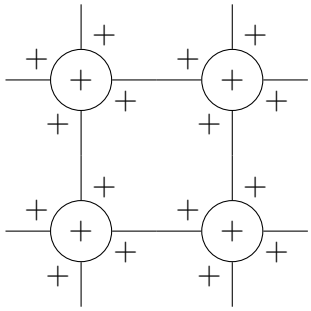
In the non-uniform case we are able to prove that intrinsic universality can be achieved in general, but that the symmetry hypothesis prohibits it while still permitting Turing universality and **P**-completeness of the prediction problem.

**Theorem 18.** *The non-uniform signed majority cellular automaton can simulate {AND,OR} circuitry in a re-usable way. It is therefore intrinsically universal by Proposition 11.*

*Proof.* The whole construction assumes that cells are in state 1 by default, which will represent the Boolean value 0. The Boolean value 1 is represented by cells in state $-1$, and these $-1$ values move along the circuit like sparse electrons. The movement of these electrons is controlled by the local sign matrix on each cell. The key point is to guarantee that cells go back to state $-1$ after the passage of these electrons.

In all the blocks gadgets given below the value on each side is simply given by a cell at a specific position. Therefore we only show the version of gadgets with all sides representing value 0. In each case we give the "impulse reponse" of the gadget showing that the desired value in computed at the output(s) at a precise time and that the gadget goes back to its initial state after the passage of electron thus ensuring re-usability.

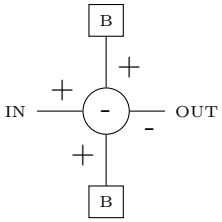
- the **background element**:



The value of the cells in this block stay unchanged whatever the context. Moreover, we can aggregate cells with all-positive signs to these blocks to pad our gadgets to a common global square shape: an all-positive signed cell stays unchanged as soon as it is connected to 2 cells that

stay unchanged. Therefore, in the following, we give only the core of the gadgets and denote by B the sides of cells that must be connect to the invariable background to ensure the correctness of the construction.

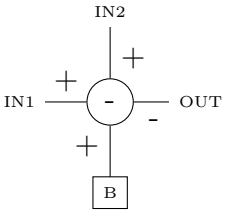
- the **wire element**:




| time | IN | OUT |
|---|---|---|
| t | 0 | 0 |
| $t+1$ | 1 | 0 |
| $t+2$ | 0 | 1 |
| $t+3$ | 0 | 0 |

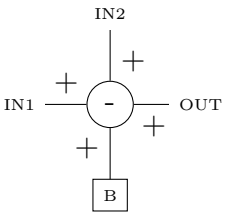
We now give the impulse response of the gadget: the B elements are supposed invariable during all the evolution, IN denotes what is received from IN by the gadget, OUT denotes what is sent to OUT from the gadget. Moreover, we assume that the cell at position OUT stays in state 1 during the evolution. As this it is not a square block of cells as required by definition 5 (in particular the B blocks are of size $2 \times 2$). However, it is straightforward to repeat it and pad with B blocks and all-+

- the **OR element**:



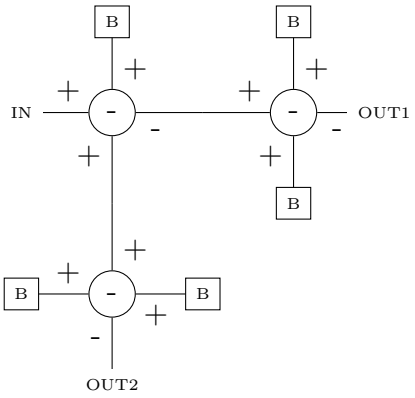

| time | IN1 | IN2 | OUT |
|---|---|---|---|
| t | 0 | 0 | 0 |
| $t+1$ | a | b | 0 |
| $t+2$ | 0 | 0 | $a \vee b$ |
| $t+3$ | 0 | 0 | 0 |

- the **AND element**:




| time | IN1 | IN2 | OUT |
|---|---|---|---|
| t | 0 | 0 | 0 |
| $t+1$ | a | b | 0 |
| $t+2$ | 0 | 0 | $a \wedge b$ |
| $t+3$ | 0 | 0 | 0 |

- the **multiply element**:




| time | IN | OUT1 | OUT2 |
|---|---|---|---|
| t | 0 | 0 | 0 |
| $t+1$ | 1 | 0 | 0 |
| $t+2$ | 0 | 0 | 0 |
| $t+3$ | 0 | 1 | 1 |
| $t+4$ | 0 | 0 | 0 |

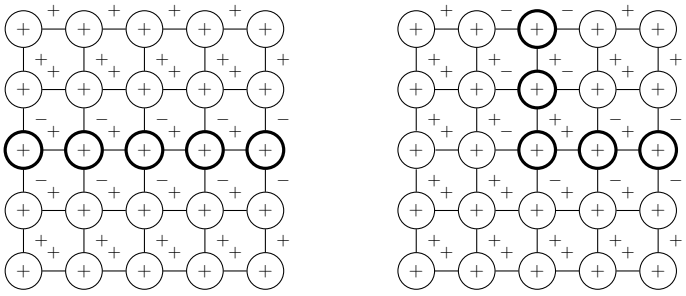
□

**Theorem 19.** *The non-uniform symmetric signed majority cellular automaton can (weakly) simulate {AND,OR,CROSS} circuitry. It is therefore Turing-universal and has a* ***P****-complete prediction problem. However, it cannot simulate {AND,OR} circuitry in a re-usable way and it is not intrinsically universal.*

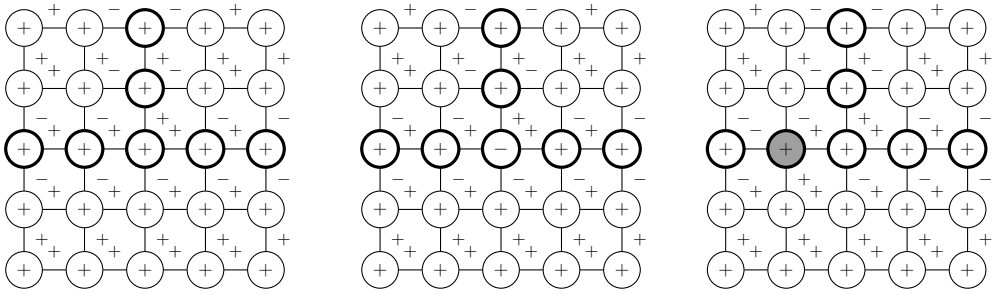
*Proof.* For the second part of the proof (impossibility to simulate re-usable monotone circuitry and non intrinsic universality) we proceed like in the proof of Theorem 14: by symmetry of the rule, Theorem 1 implies that the $f$-cycle problem is trivial when $f$ is constant equal to 3. Therefore by proposition 11 these rules cannot be intrinsically universal and cannot simulate re-usable {AND,OR} circuitry.

Figure 18 presents the elements required to apply Proposition 10. As in the asymmetric case, the CROSS gate is constructed using $(\neg west)\textsc{And}(north)$ and $(west)\textsc{And}(\neg north)$ gadgets.

□



(a) $\text{WIRE}_{n,s}$ and $\text{WIRE}_{n,e}$ gadgets. Notice that these gadgets can be turned to be used as $\text{WIRE}_{s,w}$ or $Wire_{e,n}$. Moreover, they can trivially be transformed into any other wire.



(b) AND gate, OR gate and $(\neg west)\textsc{And}(north)$ gadgets. Inputs are from north and west, and outputs to east.

Figure 18: Gadgets for the non-uniform symmetric case. The state 1 (resp. $-1$) is represented as a gray (resp. white) cell. The sign in the edges and into the nodes represent the signs of the corresponding edges in the sign matrix. Thicker nodes are highlighted because they convey the information flow. A MULTIPLY gadget can be constructed by rotating an OR gate. Like in the uniform asymmetric case, the CROSS gadget can be constructed using the $(\neg west)\textsc{And}(north)$ gate, which can be trivially rotated to be transformed into a $(west)\textsc{And}(\neg north)$ gate.

## 6. Conclusion

This paper gives new hardness and universality results for the class of signed majority cellular automata. Apart from their use in various models arising in physics, they proved to be an interesting class of cellular automata to illustrate different levels of complexity and universality related to different kinds of circuit simulation. Results are summarized in the following tables:

| **Uniform Rules** | **Symmetric** | **Antisymmetric** | **Asymmetric** |
|---|---|---|---|
| Complexity of problem Pred | $NC^1$-Hard (Theorem 14) | ? | P - Complete (Theorem 17) |
| Complexity of $f$-Cycle problem | PTIME if $f = 1$, trivial else (Theorem 1) | ? | ? |
| Universality | Not Intrinsically Universal (Theorem 14) | ? | Turing Universal (Theorem 17) |
| Maximun cycle length | 2 (Theorem 1) | at least quadratic (Theorem 15) | super-polynomial (Theorem 16) |

Table 2: Summary of the complexity results for the uniform rules, classified in Symmetric, Antisymmetric and Asymmetric.

| **Nonuniform Rules** | **Symmetric** | **Antisymmetric** | **Asymmetric** |
|---|---|---|---|
| Complexity of problem Pred | P-Complete (Theorem 19) | ? | P - Complete (Theorem 18) |
| Complexity of $f$-Cycle problem | PTIME if $f = 1$, trivial else (Theorem 1) | ? | PSPACE-hard (Theorem 18) |
| Universality | Turing Universal, but not Intrinsically Universal (Theorem 19) | ? | Intrinsically Universal (Theorem 18) |
| Maximun cycle length | 2 (Theorem 1) | at least quadratic (Theorem 15) | exponential (Theorem 18) |

Table 3: Summary of the complexity results for the non-uniform rules, classified in Symmetric, Antisymmetric and Asymmetric.

It is important to point out that C. Moore's conjecture [1], which states that the PRED for the majority rule is in **NC** (all positive weights), is not true for non-uniform signed majority (since we proved that is **P**-Complete). However, if the conjecture is true for the majority rule, then it will be for all uniform symmetric signed majority rules.

We think that this work calls for the following future research or questions:

- look for other widgets, especially in anti-symmetric rules where we have few results;
- use decreasing energy or potential techniques to prove structural results on the kind of configurations that can embed computation;
- use communication complexity to possibly prove upper bounds on some rules that would contradict the existence of a specific circuit simulation;
- what is the prediction complexity of the symmetric majority rule? is it Turing universal in at least some weak sense?

## 7. Acknowledgements

Some of the authors would like to thank CONICYT-Chile under the grants FONDECYT 1140090 (E.G.), BASAL-CMM (DIM, U. Chile) (E.G.), ECOS C12E05 (E.G. and K .P) and Conicyt Becas Chile-72130083 (P.M.).

[1] C. Moore, Majority-vote cellular automata, ising dynamics, and p-completeness, Journal of Statistical Physics 88 (1996) 795–805.

[2] D. Regnault, N. Schabanel, E. Thierry, Progresses in the analysis of stochastic 2D cellular automata: A study of asynchronous 2D minority, Theoretical Computer Science 410 (2009) 4844–4855.

[3] I. Bieche, R. Maynard, R. Rammal, J. Uhry, On the ground state of the frustration model of a spin glass by a matching method of graph theory, J. Phys. A: Math Gen. 13 (1980) 2553–2576.

[4] F. Barahona, On the computational complexity of Ising glass models, J. Phys. A: Math. Gen. 15 (1982) 3241–3253.

[5] J. Chalupa, P. Leath, G. Reich, Bootstrap percolation on a Bhete lattice, J. Phys. C:Solid State Phys. 12 (1979) L31–L35.

[6] E. Goles-Chacc, F. Fogelman-Soulie, D. Pellegrin, Decreasing energy functions as a tool for studying threshold networks, Discrete Applied Mathematics 12 (3) (1985) 261–277.

[7] R. Shingai, Maximun Period of 2-dimensional Uniform Neural Networks, Information and Control 41 (1979) 324–341.

[8] R. Shingai, The maximum period realized in 1-dimensional uniform neural networks, Trans. IECE Japan E61 (1978) 804–808.

[9] E. G. Ch., P. Montealegre-Barba, I. Todinca, The complexity of the bootstraping percolation and other problems, Theor. Comput. Sci. 504 (2013) 73–82.

[10] E. Goles, P. Montealegre, The complexity of the majority rule on planar graphs, Adv. Appl. Math. 64 (C) (2015) 111–123.

[11] B. Durand, Z. Róka, Cellular Automata: a Parallel Model, Vol. 460 of Mathematics and its Applications., Kluwer Academic Publishers, 1999, Ch. The game of life:universality revisited., pp. 51–74.

[12] E. R. Banks, Universality in cellular automata, in: Eleventh Annual Symposium on Switching and Automata Theory, IEEE, Santa Monica, California, 1970.

[13] N. Ollinger, Universalities in cellular automata, in: Handbook of Natural Computing, Springer, 2012, pp. 189–229.

[14] M. Delorme, J. Mazoyer, N. Ollinger, G. Theyssier, Bulking II: classifications of cellular automata, Theor. Comput. Sci. 412 (30) (2011) 3881–3905.

[15] E. Goles, J. Olivos, Periodic behaviour of generalized threshold functions, Discrete Mathematics 30 (2) (1980) 187 – 189. doi:http://dx.doi.org/10.1016/0012-365X(80)90121-1.

[16] E. G. Ch., Threshold networks and generalizations, in: Automata Networks, LITP Spring School on Theoretical Computer Science, France,1986, Proceedings, 1986, pp. 42–52.

[17] E. G. Ch., P. Meunier, I. Rapaport, G. Theyssier, Communication complexity and intrinsic universality in cellular automata, Theor. Comput. Sci. 412 (1-2) (2011) 2–21.

[18] C. Moore, Quasilinear cellular automata, Physica D: Nonlinear Phenomena 103 (14) (1997) 100 – 132, lattice Dynamics.

[19] M. Delorme, J. Mazoyer, N. Ollinger, G. Theyssier, Bulking I: an abstract theory of bulking, Theor. Comput. Sci. 412 (30) (2011) 3866–3880.

[20] N. Ollinger, G. Richard, Four states are enough!, Theor. Comput. Sci. 412 (1-2) (2011) 22–32.

[21] L. Boyer, G. Theyssier, On local symmetries and universality in cellular automata, in: STACS 2009 Proceedings, 2009, pp. 195–206.

[22] D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, D. Woods, The tile assembly model is intrinsically universal, in: FOCS 2012 Proceedings, 2012, pp. 302–310.

[23] E. D. Demaine, M. J. Patitz, T. A. Rogers, R. T. Schweller, S. M. Summers, D. Woods, The two-handed tile assembly model is not intrinsically universal, in: ICALP 2013 Proceedings, Part I, 2013, pp. 400–412.

[24] P. Meunier, M. J. Patitz, S. M. Summers, G. Theyssier, A. Winslow, D. Woods, Intrinsic universality in tile self-assembly requires cooperation, in: SODA 2014 Proceedings, 2014, pp. 752–771.

[25] D. A. Barrington, C.-J. Lu, P. B. Miltersen, S. Skyum, On monotone planar circuits, in: Comput. Compl., 1999.

[26] E. Allender, D. A. M. Barrington, T. Chakraborty, S. Datta, S. Roy, Planar and grid graph reachability problems, Theory of Computing Systems 45 (4) (2009) 675–723.

[27] R. Greenlaw, H. J. Hoover, W. L. Ruzzo, Limits to parallel computation: P-completeness theory (1995).

[28] T. Neary, D. Woods, P-completeness of cellular automaton rule 110, in: ICALP 2006 Proceedings, Part I, 2006, pp. 132–143.

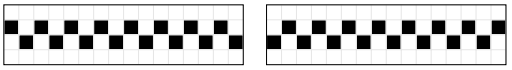
[29] E. Berlekamp, J. H. Conway, R. K. Guy, Winning ways : For your mathematical plays. 2. , games in particular (1982).

[30] D. Kozen, Theory of Computation, Texts in Computer Science, Springer, 2006.

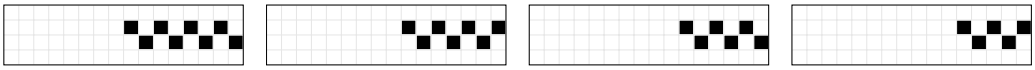
[31] G. H. Hardy, E. M. Wright, An introduction to the theory of numbers, other prints with corrections: 1962, 1965, 1968, 1971, 1975 (1960).

## Appendix A. Animations of Figure 9

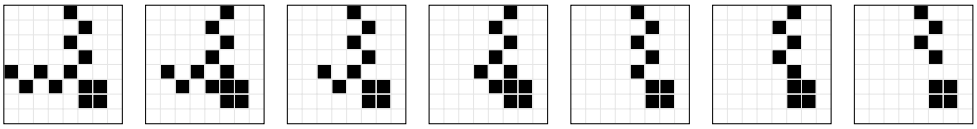
*Appendix A.1. Wire*



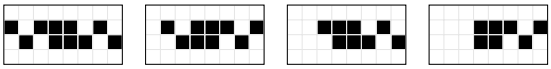
*Appendix A.2. Wire transmitting a signal*



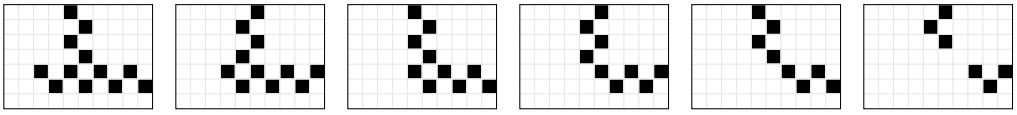
*Appendix A.3. Turn transmitting a signal*



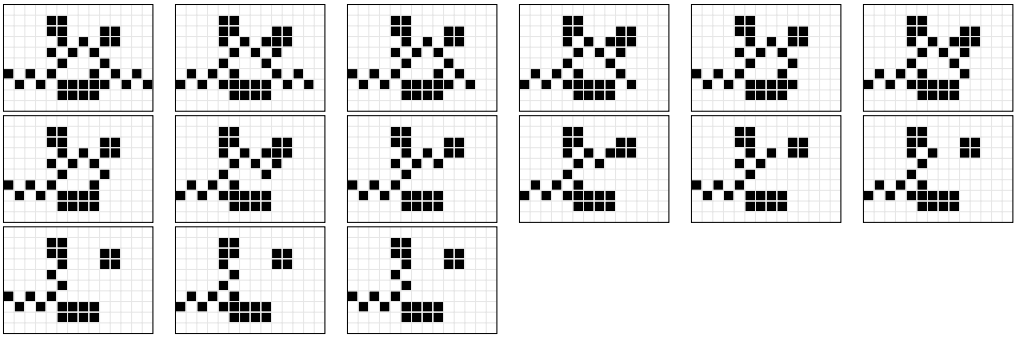
*Appendix A.4. Obstructor*



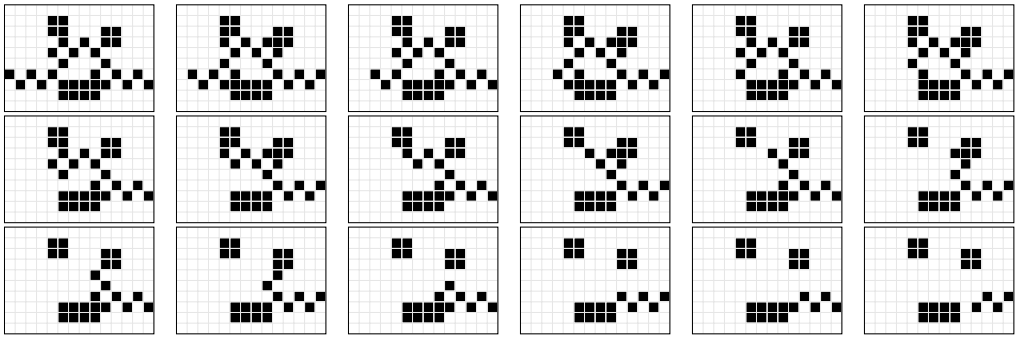
*Appendix A.5. Multiplier*



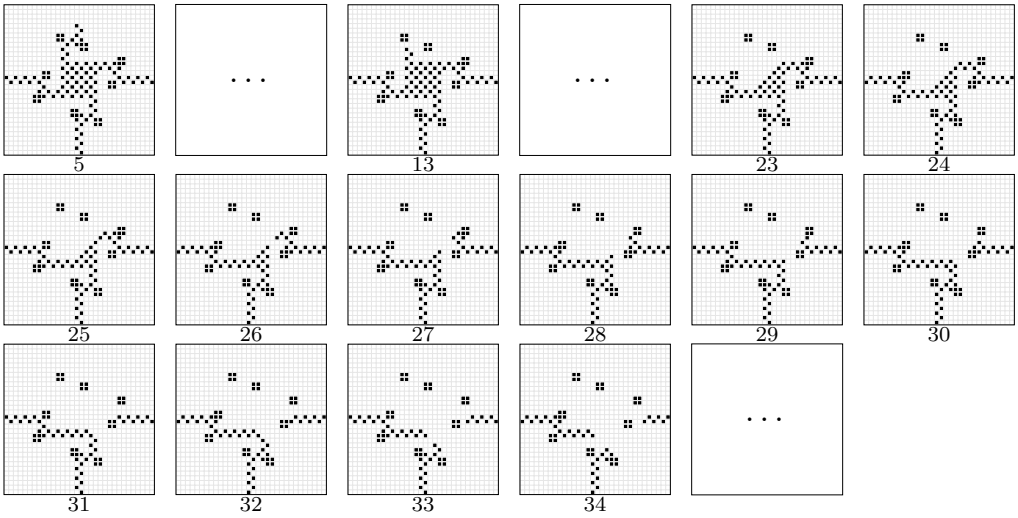
*Appendix A.6. Diode (blocking)*



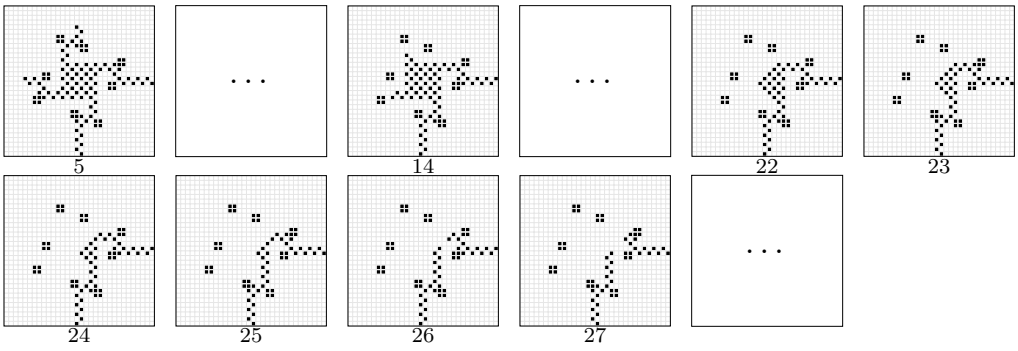
*Appendix A.7. Diode (non-blocking)*



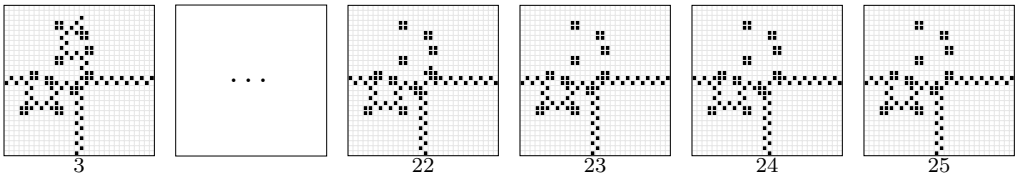
*Appendix A.8. Or (signal from the north, the west case is symmetric)*



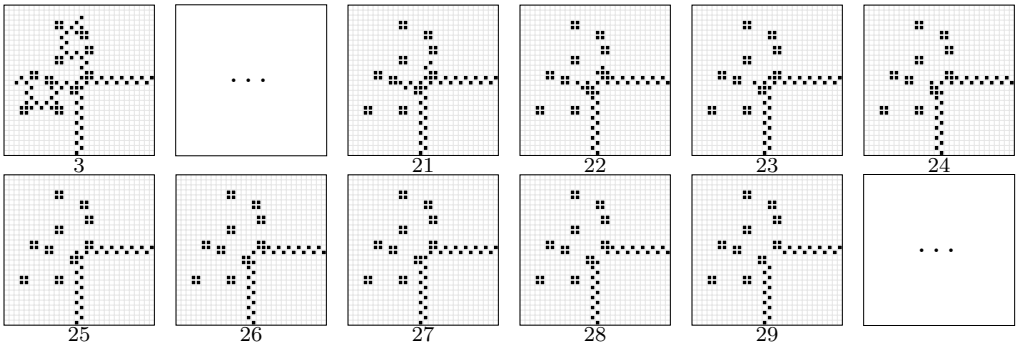

*Appendix A.9. Or (signals from the north and west)*



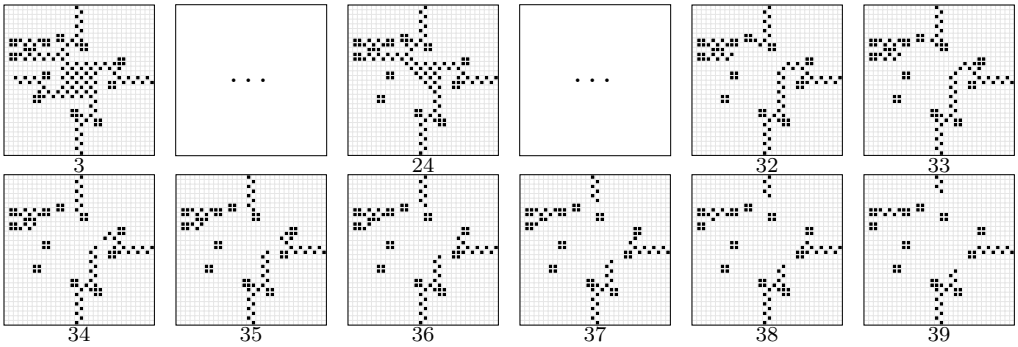
*Appendix A.10. And (signal from the north, the west case is symmetric)*




*Appendix A.11. And (signals from the north and west)*

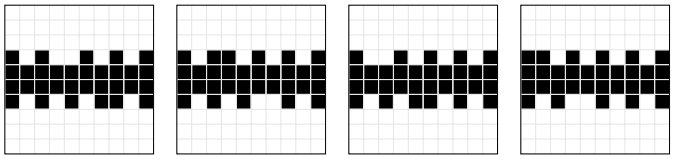



*Appendix A.12. Fusible*

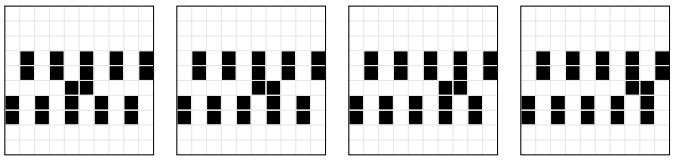



## Appendix B. Animations of Figure 16

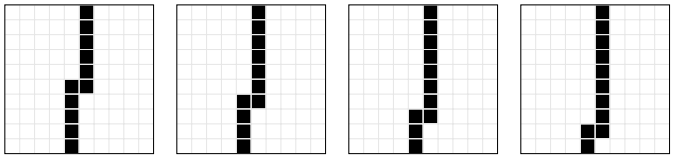
*Appendix B.1. $WIRE_{e,w}$*



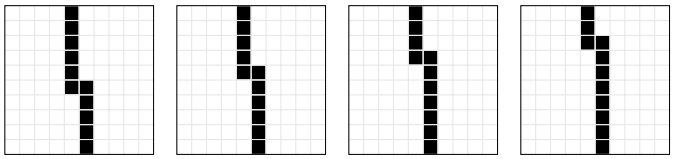
*Appendix B.2. $WIRE_{w,e}$*



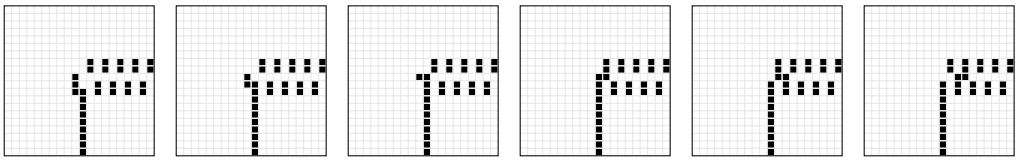
*Appendix B.3. $WIRE_{n,s}$*



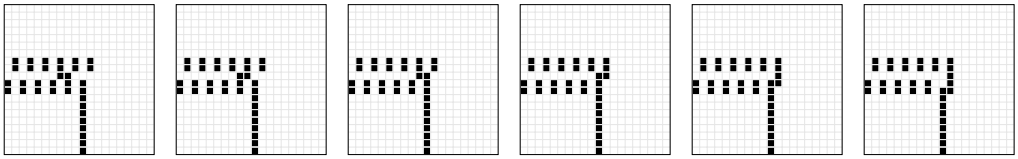
*Appendix B.4. $WIRE_{s,n}$*



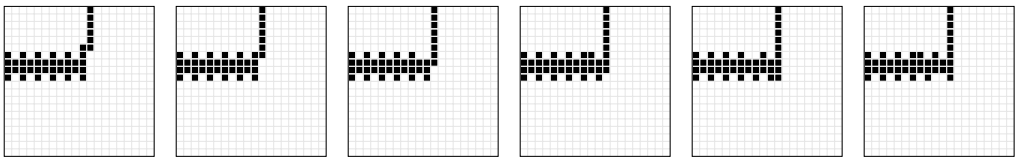
*Appendix B.5. $WIRE_{s,e}$*



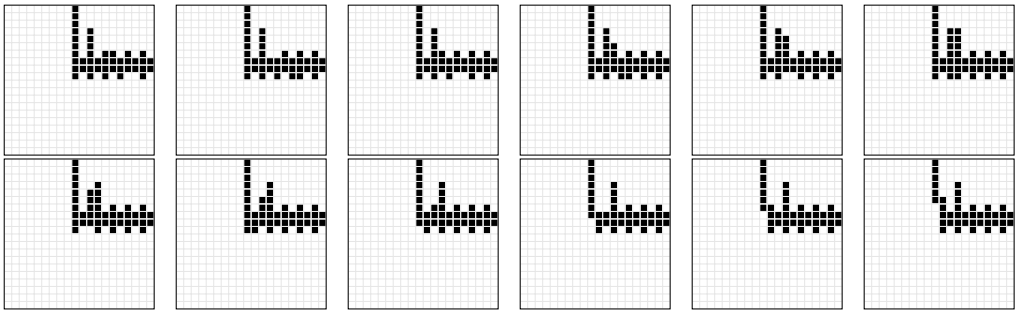
*Appendix B.6. $WIRE_{w,s}$*



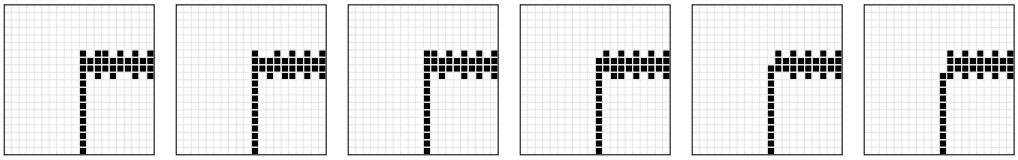
*Appendix B.7. $WIRE_{n,w}$*



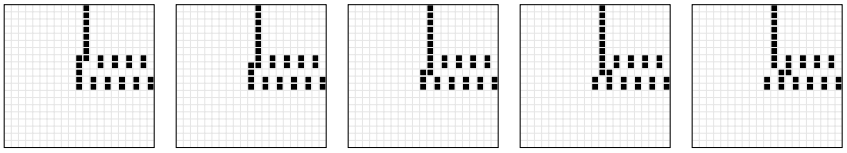
*Appendix B.8. $WIRE_{e,n}$*



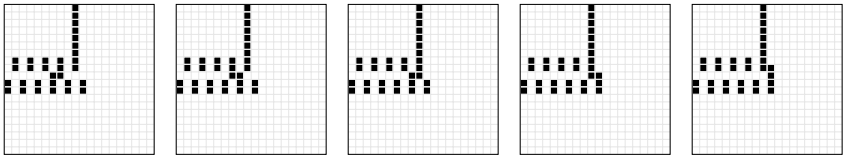
*Appendix B.9. $WIRE_{e,s}$*



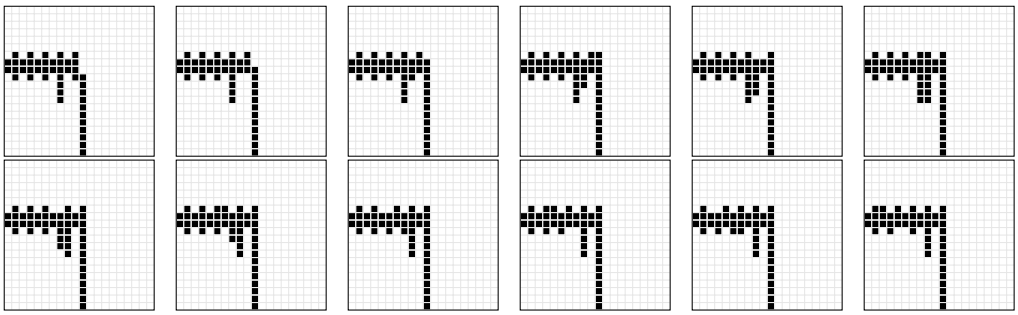
*Appendix B.10. $WIRE_{n,e}$*



*Appendix B.11. $WIRE_{w,n}$*

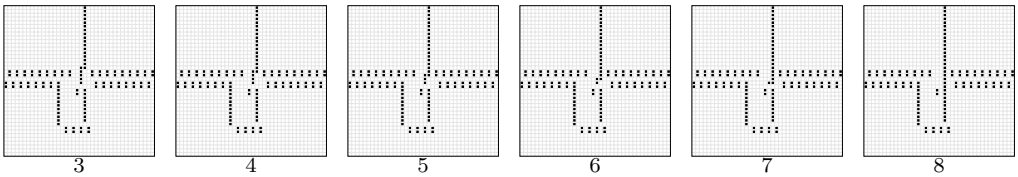


*Appendix B.12. $WIRE_{s,w}$*

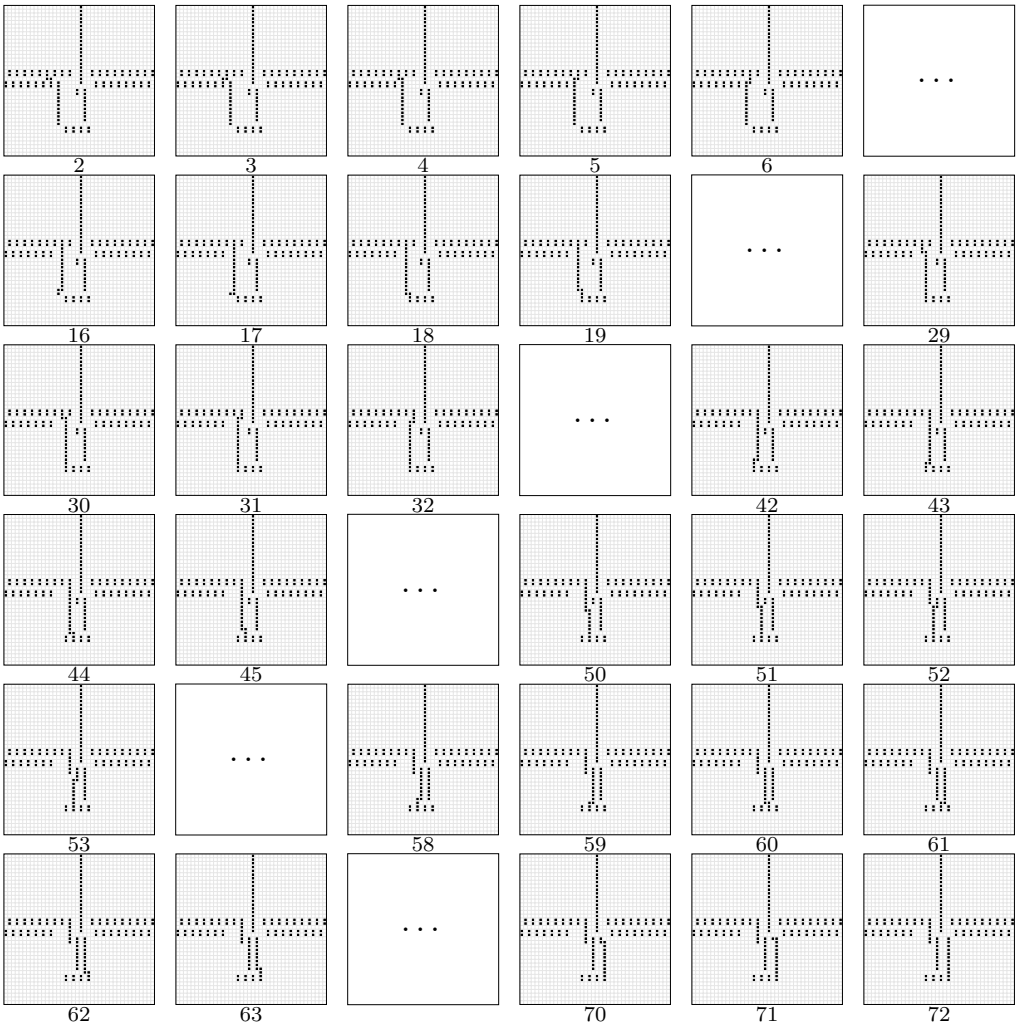


## Appendix C. Animations of Figure 17

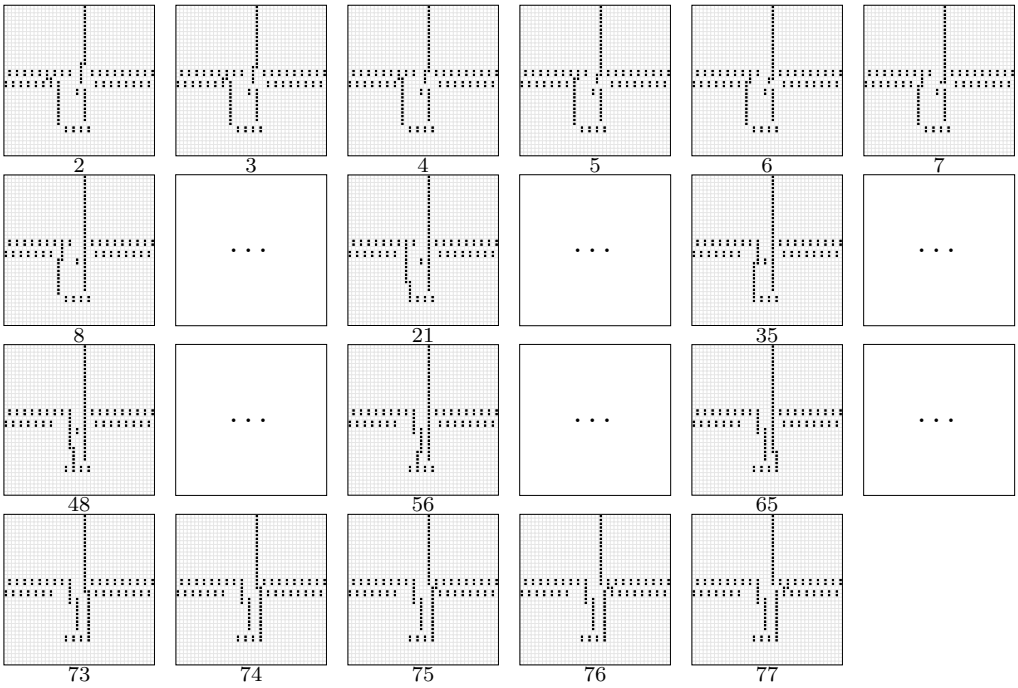
*Appendix C.1. And (signal from the north)*



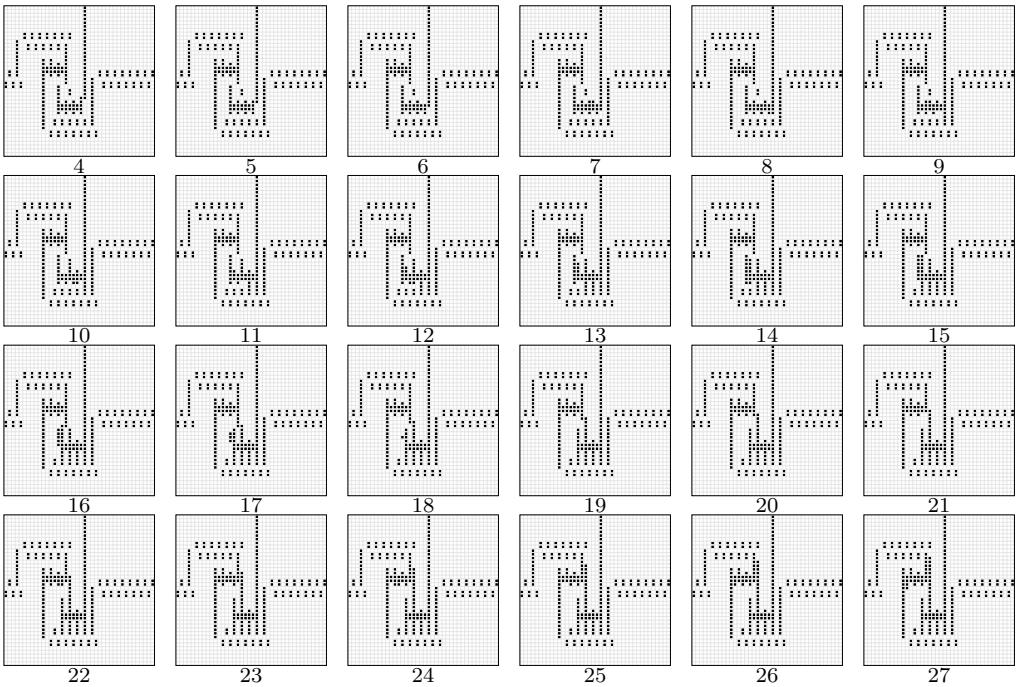

*Appendix C.2. And (signal from the west)*

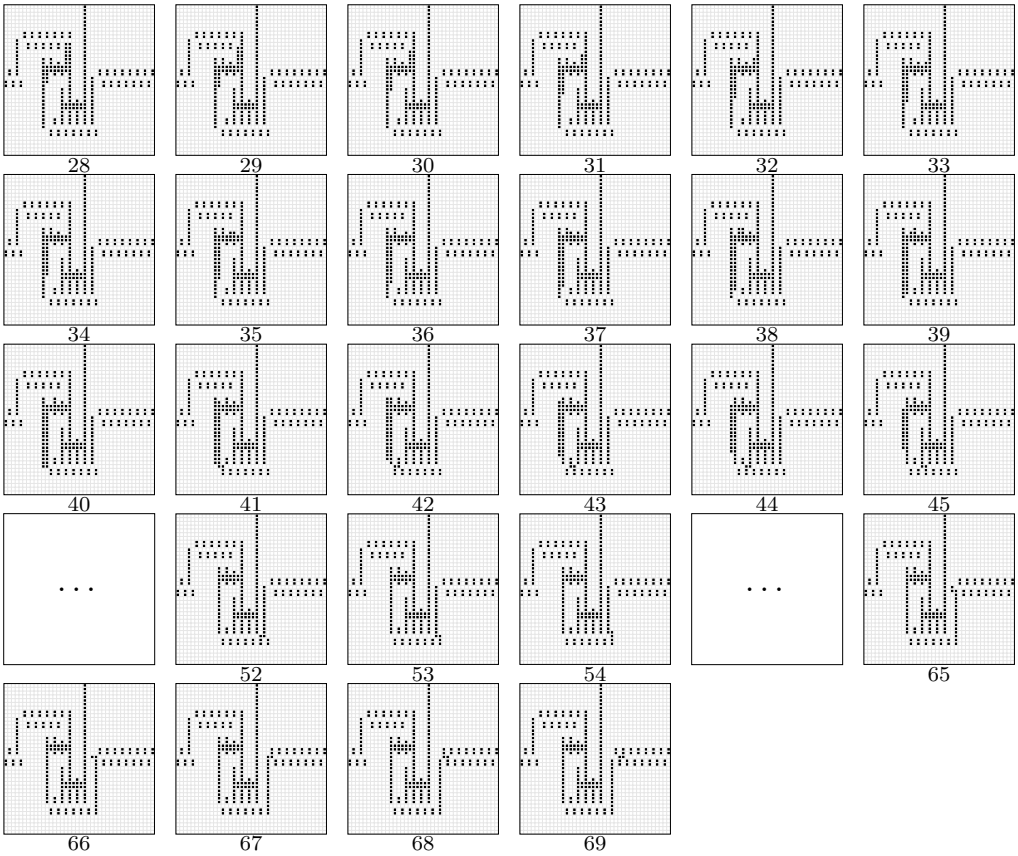


*Appendix C.3. And (signals from the north and west)*

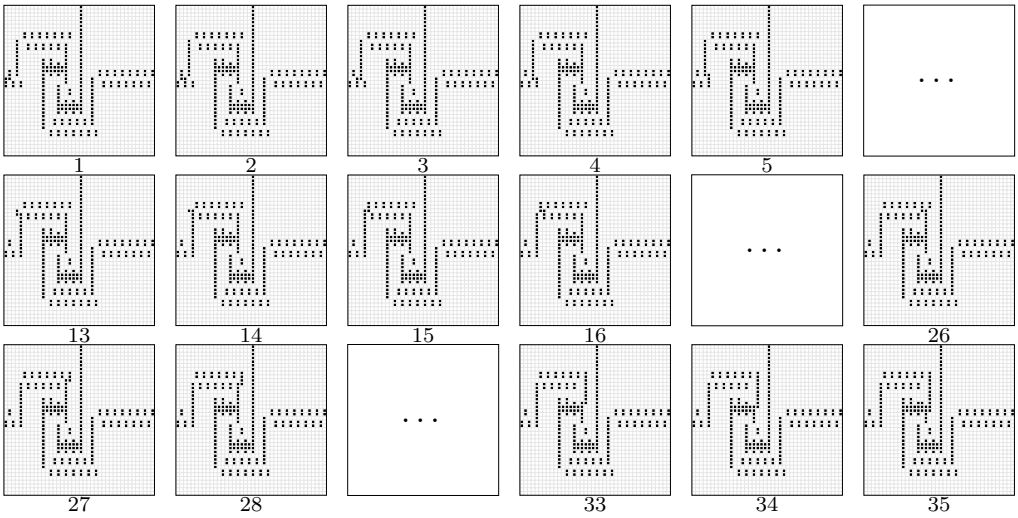


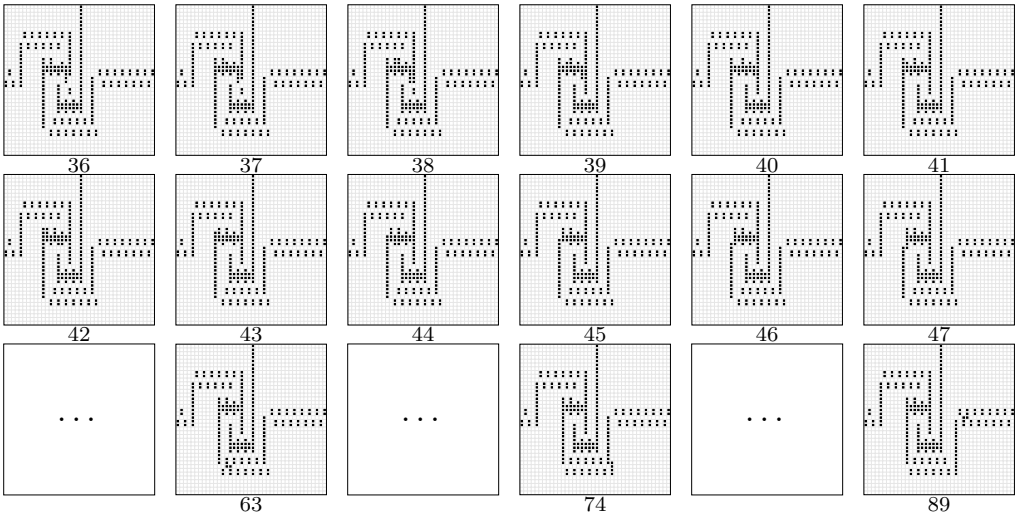

*Appendix C.4. Or (signal from the north)*



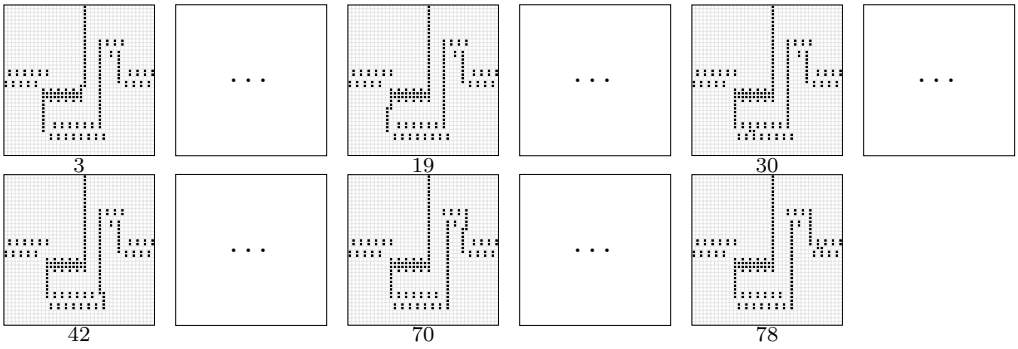



*Appendix C.5. Or (signal from the west)*

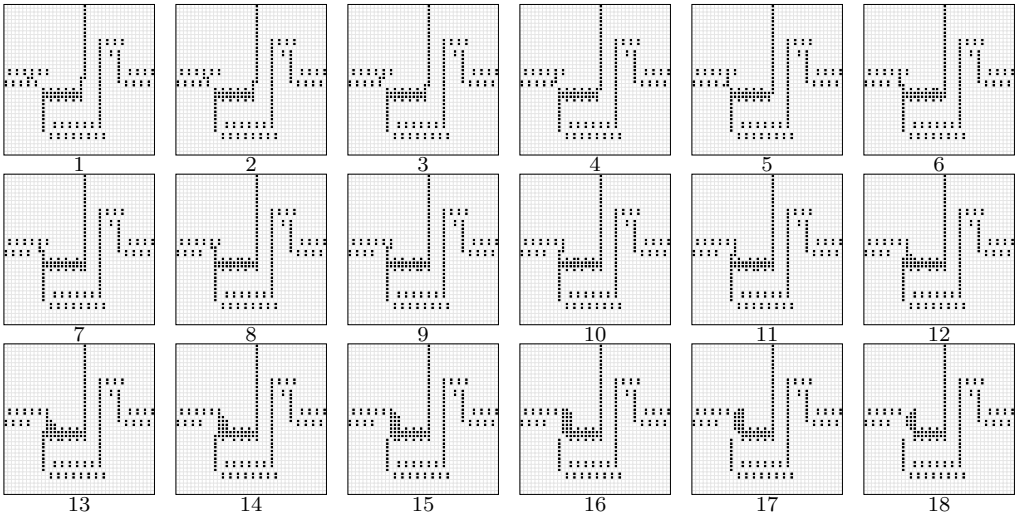


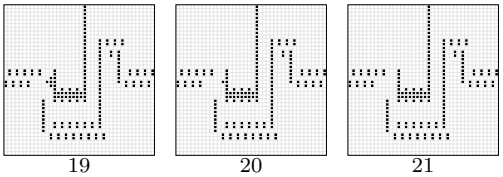



*Appendix C.6. (¬west)And(north) (signal from the north)*

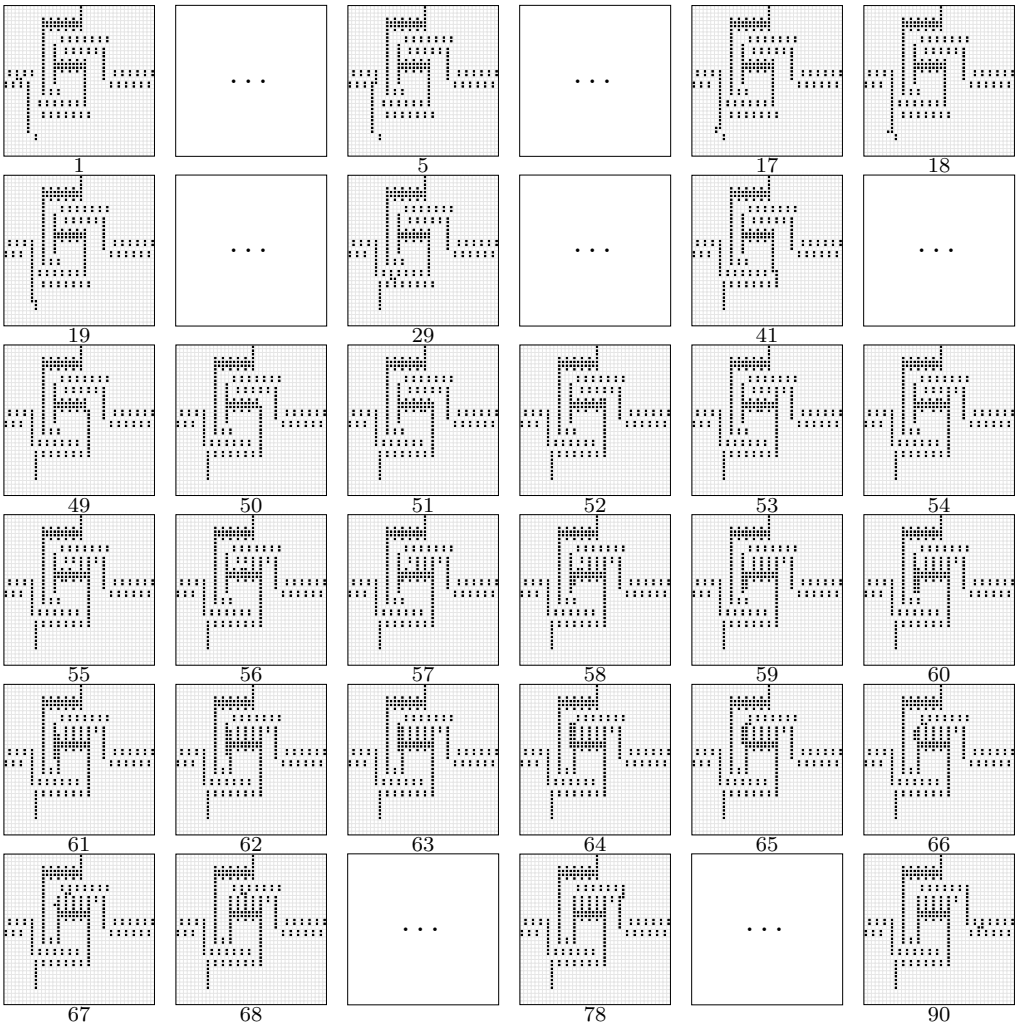



*Appendix C.7. (¬west)And(north) (signals from the north and west)*

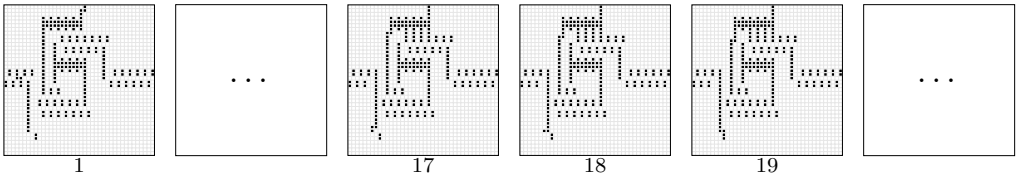


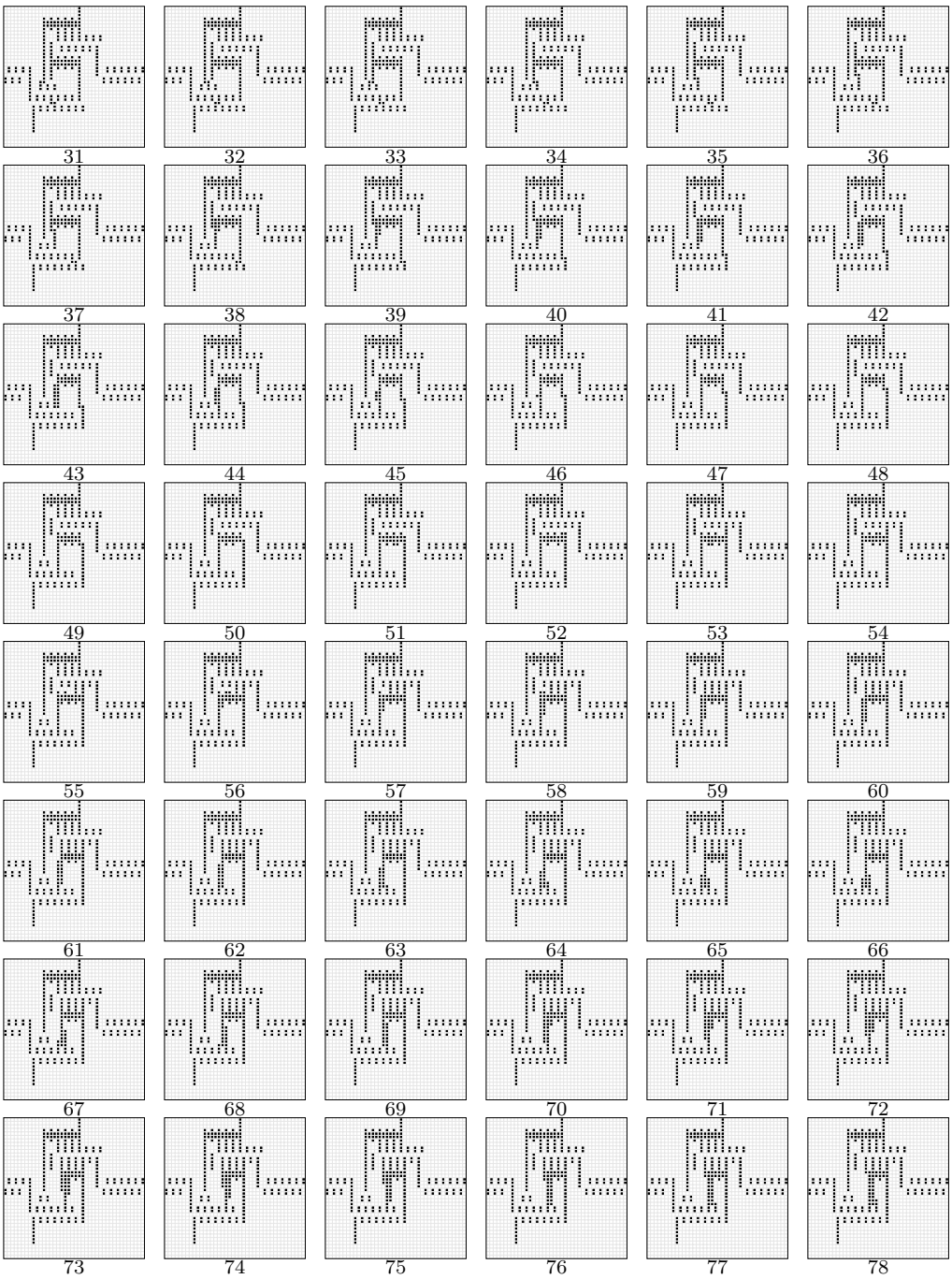


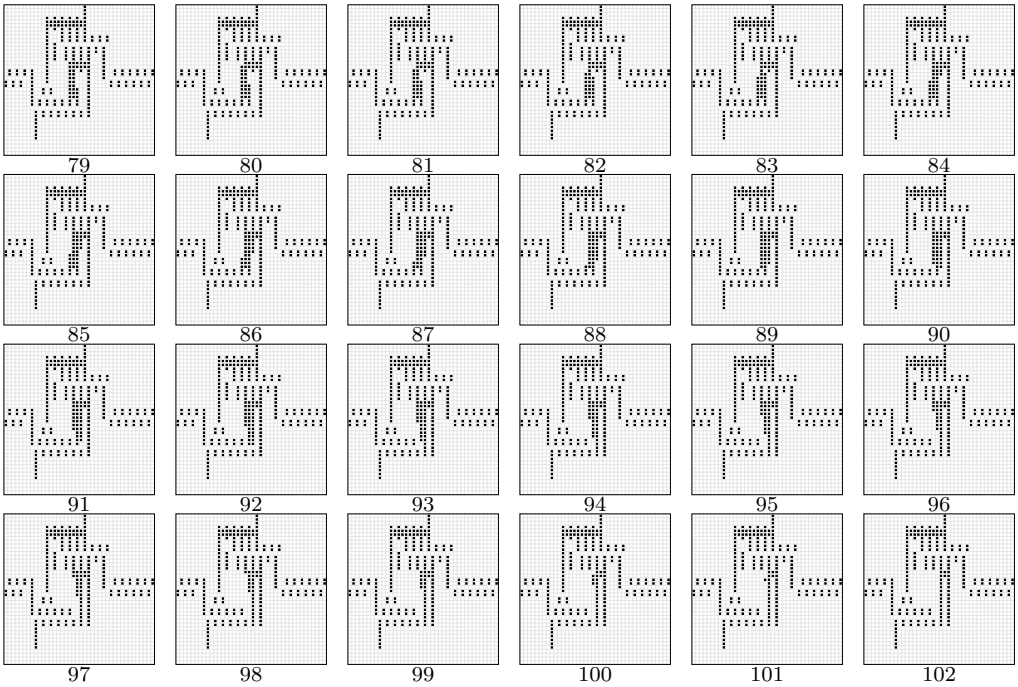

*Appendix C.8. (west)And(¬north) (signal from the west)*



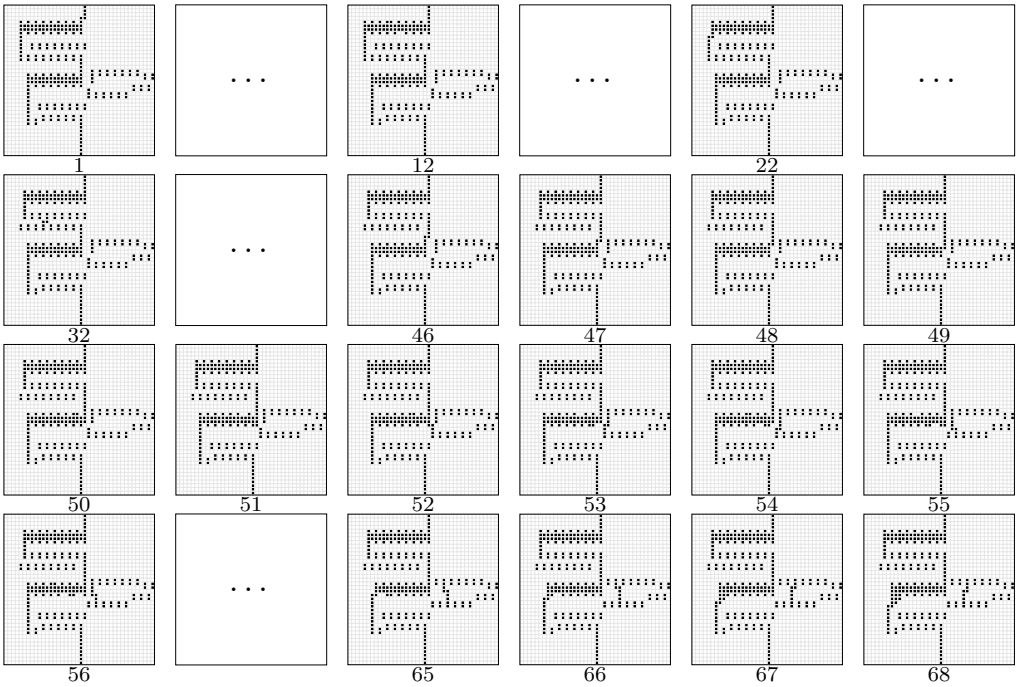

*Appendix C.9. (west)And(¬north) (signals from the north and west)*

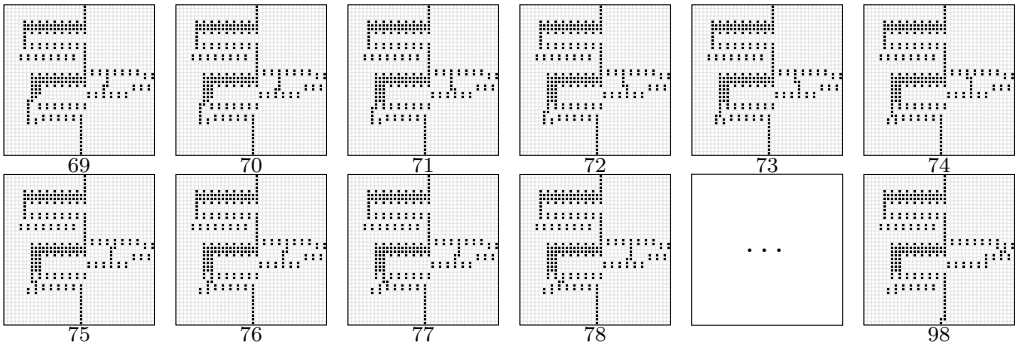



31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78




*Appendix C.10. Multiply with input from the north*






*Appendix C.11. Multiply with input from the west*