

Computing backwards with Game of Life, part 1: wires and circuits

Ville Salo and Ilkka Törmä*

Department of Mathematics and Statistics

University of Turku, Finland

{[vosalo](mailto:vosalo@utu.fi), iatorm} at utu.fi

August 18, 2023

Abstract

Conway’s Game of Life is a two-dimensional cellular automaton. As a dynamical system, it is well-known to be computationally universal, i.e. capable of simulating an arbitrary Turing machine. We show that in a sense taking a single backwards step of Game of Life is a computationally universal process, by constructing patterns whose preimage computation encodes an arbitrary circuit-satisfaction problem, or (equivalently) any tiling problem. As a corollary, we obtain for example that the set of orphans is coNP-complete, exhibit a 6210×37800 -periodic configuration that admits a preimage but no periodic one, and prove that the existence of a preimage for a periodic point is undecidable. Our constructions were obtained by a combination of computer searches and manual design.

1 Introduction

Conway’s Game of Life, designed by John Conway and popularized by Martin Gardner in [6], is a two-dimensional cellular automaton. Specifically, it is the function $g : \{0, 1\}^{\mathbb{Z}^2} \rightarrow \{0, 1\}^{\mathbb{Z}^2}$ defined by

$$g(x)_{\vec{v}} = \begin{cases} 1, & \text{if } x_{\vec{v}} = 0 \text{ and } \sum x|_{\vec{v} + [-1,1]^2} = 3, \\ 1, & \text{if } x_{\vec{v}} = 1 \text{ and } \sum x|_{\vec{v} + [-1,1]^2} \in \{3, 4\}, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

If configurations $x \in \{0, 1\}^{\mathbb{Z}^2}$ are interpreted as infinite grids of live/black/occupied cells (denoted by 1) and dead/white/empty cells (denoted by 0), then the rule can be interpreted as saying that a new live cell is born at a dead cell with exactly three live neighbors, and a live cell will stay alive if and only if it has two or three live neighbors.

Game of Life, being a function from a topological (specifically, Cantor) space to itself, can be interpreted as a dynamical system. As the space is very combinatorial, it can also naturally be interpreted as a computational device, and

*Ilkka Törmä was supported by the Academy of Finland under grant 346566.

these two points of view are strongly intertwined. It is known that Game of Life is intrinsically universal [4], i.e. its subsystems can simulate any cellular automaton. It is also known that starting from a finite-support configuration (one with finitely many live cells) it can simulate the behavior of a Turing machine on a finite configuration [16].¹ Thus, it is natural to say that the dynamics of Game of Life is computationally universal. See [11, 12] for more information on building patterns in Game of Life with interesting dynamics, which is a rather vast field of science on its own.

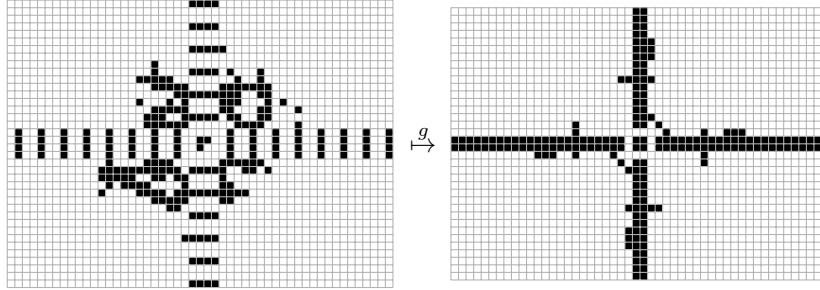
In addition to its dynamics, Game of Life can also thought of as a *block map*, i.e. a continuous shift-commuting map between two subshifts $X = \{0, 1\}^{\mathbb{Z}^2}$ and $Y = \{0, 1\}^{\mathbb{Z}^2}$, which furthermore happen to be the same. In this point of view, our attention moves from iteration to “one-step” problems. For example, we may ask about the image of Game of Life $g(\{0, 1\}^{\mathbb{Z}^2})$ (in symbolic dynamics jargon, this is a particular *sofic shift*), about its fixed points (in symbolic dynamics jargon, this is a particular *subshift of finite type*, or *SFT*), or about the preimage $g^{-1}(0^{\mathbb{Z}^2})$ of the completely empty configuration (this is another *SFT*).

Some previous results in this direction are that $g^{-1}(0^{\mathbb{Z}^2})$ has dense semilinear points but does not have dense periodic points [18], and the maximal density of 1s in the subshift of fixed points is exactly $1/2$ [5]. There are also many open problems about such subshifts, for example, it is open whether the subshift of fixed points has dense finite-support configurations (this is called the still life finitization problem).

One may also consider the study of temporally periodic points as being about one-step dynamics (seeing higher powers of Game of Life as block maps). An interesting problem is *omniperiodicity*, or whether Game of Life admits finite-support configurations of all periods. It was recently solved in the positive [15], ending a decades-long collaborative project of the Game of Life research community. It remains open what other properties the periodic-point subshifts have; for example, like with the fixed-point subshift, it is natural to ask whether finite-support points are dense.

In this paper, we prove that Game of Life is “universal as a block map” (in a rather technical sense discussed in Section 7). In particular, we can argue that it is computationally universal even without iteration. Specifically, we show that in carefully engineered situations, computing the preimage for a configuration is roughly equivalent to the problem of finding a satisfying assignment to a Boolean circuit. We construct “gadgets” (square patterns $\{0, 1\}^{n \times n}$) that represent logical gates, such that the preimages of circuits built from these gadgets are in correspondence with the satisfying assignments of the corresponding abstract circuit. For example, we show below a “wire-crossing” gadget, which allows two signals to pass through it without interacting, and a corresponding preimage where signals cross:

¹Adam Goucher obtains a simultaneous proof of both claims by proving intrinsic universality with a quiescent state simulated by 0-blocks; see [11, Chapter 12].



The black lines of width 2 on the sides are used as (and are also called) *wires*, and the signal they carry is the phase modulo 3 of the repeated length-4 segments of live cells in the preimage – it turns out that this simple pattern propagates through a wire. We also present gadgets for tasks like turning wires, inverting the signal they carry, and logical operations, as well as more technical gadgets that introduce the signal into wires and change the phase of the signal traveling on them.

Our general construction allows us to turn the problem of satisfying a circuit into a preimage computation problem, giving us the following results:

- given a finite-support configuration, it is NP-complete whether it admits a preimage under the Game of Life (Theorem 5; NP was proved in [18]),
- given a finite pattern, it is NP-complete whether it appears in the image subshift of Game of life, equivalently whether it is an orphan is co-NP-complete (Theorem 6).

We can also simulate tilings of the two-dimensional plane by drawing circuits that check the color constraints of a set of Wang tiles. An immediate, but perhaps striking, corollary is that there exists a periodic configuration that has a preimage, but has no periodic preimage (Theorem 2). Using a small aperiodic tile set, such as the 11-tile Jeandel-Rao set [10], we can make the periods small enough (6210×37800) to fit comfortably in computer memory. In the ancillary files, the reader can find an explicit presentation compatible with the Golly simulator [17]; screenshots are shown in Figure 1.

On the theoretical side, we obtain the following results:

- there exists a periodic configuration that has a preimage, but has no recursive (= computable) preimage (Theorem 10),
- there exists a periodic configuration that has a preimage, but every preimage has $\Omega(n)$ Kolmogorov complexity in every $n \times n$ -pattern (Theorem 11),
- given a periodic configuration, it is undecidable whether it admits a preimage (Theorem 8),
- given a periodic configuration that has a preimage, it is undecidable whether it has periodic preimage (Theorem 9)

We emphasize that no real work is needed to establish these corollaries – we are simply transporting some well-known results from tiling theory into Game of Life using our circuit technology.

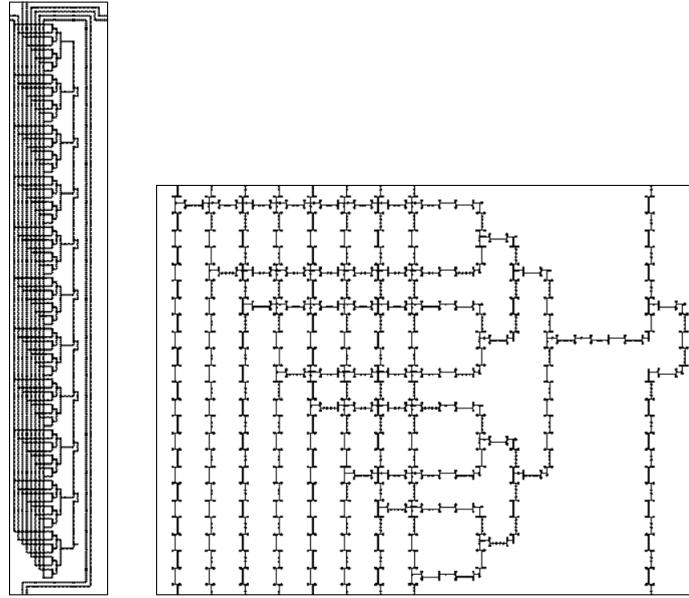


Figure 1: On the left, the fundamental domain (a 6210×37800 rectangle) of a periodic configuration whose preimages factor onto scaled tilings of the Jeandel-Rao tile set (thus are aperiodic). On the right, a subpattern roughly corresponding to a single tile type.

The technical framework where we prove most of our results is adapted from the idea of intrinsic universality [14] in cellular automata theory. The basic idea is that a universal block map ϕ can simulate any other block map ψ , when symbols of (the codomain of) ψ are replaced by suitable patterns (in the codomain) of ϕ . We in fact introduce three notions of universality for block maps, namely weak, semiweak and strong universality. We develop a basic theory of these notions, and show that a certain block map taking satisfied logical circuits to their underlying circuits (all encoded as subshifts of finite type) is strongly universal.

We prove that Game of Life semiweakly simulates the circuit system, and conclude that it is semiweakly universal (Theorem 4), which suffices to prove all the results above (indeed, all but Theorem 9 follow from weak universality). We leave open whether Game of Life is strongly universal. This would imply, for example, that it is undecidable whether a periodic configuration that admits some preimage also admits an aperiodic preimage, which for now we cannot conclude.

All of our gadgets were found using SAT-solvers to find preimages (and prove the lack of particular types of preimages) for patterns, and then looking for suitable patterns using various types of search methods, in some cases accompanied by a bit of manual experimentation. The results are conditional to SAT-solvers' claims about the gadgets being correct. We have not been able to verify any but the most obvious claims with straightforward search algorithms. Information on the implementation and testing can be found in Section 9.

Our paper is by no means the first time “NP-completeness gadgets” are found by computer, see references in [7].

While our results are particular to a single step of Game of Life, the method of finding constrained preimages by computer searcher accompanied by SAT-solvers is in principle applicable to any cellular automaton in at least two dimensions (and any fixed number of iterations of one). However, the methods do not readily extend to beyond a fixed number of iterations, and even then the amount of computational resources needed grows rapidly with the radius of the CA.

Of particular interest would be a general method for performing *long-term* backwards computation in Game of Life. So far, we do not have such tools. Namely, while we show that in a single step backwards in time, a configuration can enforce (even infinitary) computation on every preimage, there is nevertheless a lot of freedom in the choice of the preimage, and thus we have no control on the second-order preimage.

Question 1. *Can similar results be proved for higher powers of Game of Life?*

Question 2. *Are there Game of Life configurations such that all of their preimage chains perform universal computation backwards in time?*

2 Definitions

Fix a dimension $d \geq 1$, and let A be a finite set, called a *state set* or *alphabet*. A d -dimensional *pattern* over A is a mapping $P : D \rightarrow A$ for a *domain* $D = D(P) \subseteq \mathbb{Z}^d$. The symbol at position $\vec{v} \in D$ in P is denoted $P_{\vec{v}}$. A *finite pattern* is a pattern with a finite domain, and a *configuration* is a pattern whose domain is \mathbb{Z}^d . Configurations are denoted by small letters x, y, z, \dots , and the set $A^{\mathbb{Z}^d}$ of all configurations is the d -dimensional *full shift* over A . If A contains a distinguished “zero symbol” (which is either the literal 0 or otherwise clear from context), the *support* of a configuration x is the set $\text{supp}(x) = \{\vec{v} \in \mathbb{Z}^d \mid x_{\vec{v}} = 0\}$. We also use the obvious analog for finite patterns. The *shift maps* $\sigma^{\vec{v}} : A^{\mathbb{Z}^d} \rightarrow A^{\mathbb{Z}^d}$ for $\vec{v} \in \mathbb{Z}^d$ are defined by $\sigma^{\vec{v}}(x)_{\vec{w}} = x_{\vec{w} + \vec{v}}$. They can also be applied to patterns: $\sigma^{\vec{v}}(P)$ is the pattern $Q \in A^{D(P) - \vec{v}}$ with $Q_{\vec{w} - \vec{v}} = P_{\vec{w}}$ for all $\vec{w} \in D(P)$. If $d = 1$, we denote the *left shift* by $\sigma = \sigma^1$. A pattern P occurs in a configuration x , denoted $P \sqsubset x$, if there exists $\vec{v} \in \mathbb{Z}^d$ with $\sigma^{\vec{v}}(x)|_{D(P)} = P$.

Every set F of finite patterns over A defines a *subshift* $X \subset A^{\mathbb{Z}^d}$ as the set of configurations where no element of F occurs. Symbolically,

$$X = \{x \in A^{\mathbb{Z}^d} \mid \forall P \in F : P \not\sqsubset x\}.$$

If F is finite, then X is a *shift of finite type* (SFT).

One way to define two-dimensional SFTs is by *Wang tile sets*. A *Wang tile* over a finite set C of *edge colors* is a 4-tuple $t = (t_E, t_N, t_W, t_S) \in C^4$. A set $T \subseteq C^4$ of Wang tiles defines an SFT $X_T \subseteq T^{\mathbb{Z}^2}$ using the forbidden patterns

$$t^1 t^2 \quad \text{and} \quad \frac{t^3}{t^4}$$

for all $t^1, t^2, t^3, t^4 \in T$ with $(t^1)_E \neq (t^2)_W$ and $(t^3)_S \neq (t^4)_N$, respectively. A Wang tile can be visualized as a unit square whose east, north, west and south

edges are labeled with the four colors in this order. A configuration of Wang tiles is in X_T if the edge colors of all adjacent tiles match.

Let $X \subseteq A^{\mathbb{Z}^d}$ and $Y \subseteq B^{\mathbb{Z}^d}$ be subshifts. A *block map* is a function $f : X \rightarrow Y$ that admits a finite *neighborhood* $N \subset \mathbb{Z}^d$ and a *local rule* $F : A^N \rightarrow B$ such that $f(x)_{\vec{v}} = F(\sigma^{\vec{v}}(x)|_N)$ holds for all $x \in X$ and $\vec{v} \in \mathbb{Z}^d$. The image $f(X)$ of a block map is always a subshift, and if X is an SFT, then $f(X)$ is called a *sofic shift*. A *cellular automaton* is a block map from a full shift to itself. Once the neighborhood N and local rule F of a block map $f : X \rightarrow Y$ have been fixed, we define the preimage of a pattern $P \sqsubset Y$ as

$$f^{-1}(P) = \{Q \in A^{D(P)+N} \mid Q \sqsubset X, \forall \vec{v} \in D(P) : F(\sigma^{\vec{v}}(Q)|_N) = P_{\vec{v}}\}.$$

Note that the domain of $f^{-1}(P)$ is usually larger than that of P .

The Game of Life $g : \{0,1\}^{\mathbb{Z}^2} \rightarrow \{0,1\}^{\mathbb{Z}^2}$ is the block map defined in (1), and it admits the neighborhood $N = [-1,1]^2$. We will not use the letter g for any other purpose.

A *section* of a block map $f : X \rightarrow Y$ is a block map $h : Y \rightarrow X$ such that $f(h(y)) = y$ for all $y \in Y$. We say a block map *splits* or *is split* if it admits a section. If a block map $f : X \rightarrow Y$ is bijective, it is called a (*topological*) *conjugacy*, and then X and Y are *conjugate* subshifts. The inverse function of a conjugacy is always a block map.

Let A and B be alphabets and $R = [0, n_1 - 1] \times \cdots \times [0, n_d - 1]$ a d -dimensional hyperrectangle. A *substitution* of shape R is a mapping $\tau : A \rightarrow B^R$. It can be extended to a mapping $\tau : A^{\mathbb{Z}^d} \rightarrow B^{\mathbb{Z}^d}$ by $\tau(x)_{\vec{v}} = \tau(\sigma^{\vec{w}}(x)_{\vec{0}})_{\vec{u}}$, where $\vec{u} \in R$ and $\vec{w} \in \prod_{i=1}^d n_i \mathbb{Z}$ are the unique vectors that satisfy $\vec{u} + \vec{w} = \vec{v}$. Intuitively, we apply the “local” map τ to every symbol of x , obtain a configuration of R -shaped patterns over B , and then stitch them together to obtain a configuration over B .

For our computational complexity results we need to fix encodings of various objects into binary words. Each alphabet A is assumed to come with a binary encoding $b : A \rightarrow \{0,1\}^+$ that is unambiguous which is injective as a morphism, in the sense that the mapping from words $a_1 \cdots a_n \in A^*$ of arbitrarily length to concatenations $b(a_1) \cdots b(a_n)$ is injective. For hyperrectangles of the form $R = [-n_1, n_1] \times \cdots \times [-n_d, n_d]$, we encode patterns $P \in A^R$ as $0^{n_1} 1 \cdots 0^{n_d} 1 u$, where u is the concatenation of $b(P_{\vec{v}})$ for $\vec{v} \in R$ in lexicographical order. A finite-support configuration $x \in A^{\mathbb{Z}^d}$ is encoded identically to the pattern $x|_R$, where $R \subset \mathbb{Z}^d$ is the smallest hyperrectangle of the above form containing $\text{supp}(x)$. For a fixed universal Turing machine U , the *Kolmogorov complexity* of a word $w \in \{0,1\}^*$ is the length of the shortest word $v \in \{0,1\}^*$ such that U computes w when given v as input. The Kolmogorov complexity of a hyperrectangular pattern P is defined as the Kolmogorov complexity of the concatenation of $b(P_{\vec{v}})$ for $\vec{v} \in D(P)$ in lexicographical order. Note that the shape and position of the pattern are not encoded.

3 Wires and gadgets

In our constructions we use wires, which are a type of *self-propagating pattern*, namely a pattern P associated to a vector $\vec{p} \neq \vec{0}$ with the property that whenever the f -preimage x of a suitable configuration y contains an occurrence of P at

some coordinate \vec{v} , then it contains another occurrence at $\vec{v} + \vec{p}$. In our case, $\vec{p} \in \{(0, \pm 3), (\pm 3, 0)\}$, and y is suitable if it resembles the all-1 configuration near the two occurrences of P . If y is suitable near $\vec{v} + n\vec{p}$ for $n = 2, 3, \dots$, then the pattern must occur in x at these coordinates as well.

3.1 Wires

A *vertical wire* is a pattern of shape $2 \times n$ consisting of 1-symbols, and a *horizontal wire* is a pattern of shape $n \times 2$ consisting of 1-symbols. Information is transmitted along a wire in the form of a 3-periodic pattern. The *vertical wire signals* are the three 4×2 patterns

$$W_0 = \begin{matrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{matrix} \quad W_1 = \begin{matrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix} \quad W_2 = \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Rotating them by 90 degrees counterclockwise results in the *horizontal wire signals*.

Consider a configuration $x \in \{0, 1\}^{\mathbb{Z}^2}$ and $y = g(x)$. If $x|_{[0,3] \times [0,1]} = W_i$ and $y|_{[1,2]^2} = \begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}$, then a short case analysis reveals that $x|_{[0,3] \times [1,2]} = W_{i+1}$, where the index is taken modulo 3. For example, in the case $i = 0$ both $x_{(1,1)}$ and $x_{(2,1)}$ must have three 1-symbols in their neighborhood, which implies $x|_{[0,3] \times \{2\}} = 1111$. (One may also check that already a single 1111-row in the preimage will force this periodic structure on the preimage.) Thus each W_i is self-propagating along a vertical wire in both directions $\vec{p} = (0, \pm 3)$, and the analogous claim holds for the horizontal signals and wires. We use the phase of the signal to transmit information within x for a fixed image y .

Note that the above property holds no matter what the 1s are surrounded with. Typically we will surround the wires with 0s, and use a line of 1s of thickness two as the wire. In Figure 2 we show a wire (the red color means the same as black, i.e. a live cell), and the three *charged* preimages, i.e. ones carrying a signal. We only show the part of the preimage that is forced by the presence of a horizontal wire signal (in any part of the wire).

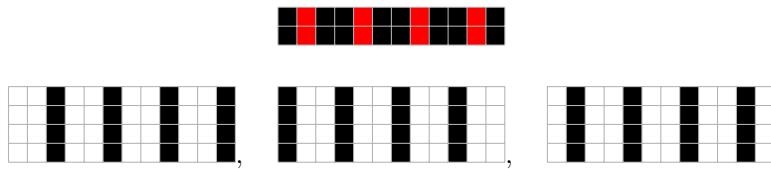


Figure 2: A wire and its three charged preimages.

The red color does not have an immediate mathematical meaning, and one can read it as black in all figures. However, it will have an important role in reading our gadgets: we will use two-valued logic throughout, so we will always charge our wires so that the wire carries a signal in one of two phases. The red color is used to mark which phase is unused, and specifically we use the red color for the cells whose preimage should always contain zeroes independently of the bit carried by the signal phase. In Figure 2, the unused phase, according to how we have colored the wire, is the rightmost one.

4 The basic gadgets

In this section, we introduce the so-called *basic gadgets* which “manipulate” signals traversing a wire. Again, a wire is *charged* if (in a particular preimage or family of preimages) its preimage consists of the W_i -patterns. As mentioned in the introduction, we verify such properties of our gadgets by SAT solvers, and we have not checked (all of) them through other means; thus, this section simply *describes* the properties of the gadgets, and does not prove them. See Section 9 for details on how we found and verified the gadgets.

Formally, a rectangle-shaped pattern can be thought of as defining a relation on the cells on the boundary of its preimages. Our basic gadgets are all rectangular patterns, with some common properties, which will allow us to do computer science with them. They all have the following properties in common:

1. Some cells on the boundary of the gadget are part of wires pointing out of the domain. Each of the four borders will have at most one wire.
2. The gadget defines some relation on the possible phases of signals on the wires, by admitting some combinations in a preimage and forbidding others.
3. Each gadget carries the information, for each of the boundary wires, of which two phases of a signal denote 0 and 1. They are called the *operating phases* of the gadget.
4. Each realizable combination of operating phases on the boundary wires can also be realized with zeros everywhere else on the boundary.

The last item is a key property, as it allows us to worry only about the wires: the relations on the wires already force whatever computation we want to happen, and for any such computation, we know there is a legal preimage, because we can simply write zeroes everywhere on the boundaries and stitch the gadgets together along them. We now give precise descriptions of our gadgets and their behavior.

In a vertical wire, we will always interpret a charged wire as containing the logical value 0 (resp. 1) if, out of the two possible adjacent positions where the 1s could occur, they occur on the north (resp. south) side, i.e. in the preimage of the non-red cells we have a copy of W_0 (resp. W_1). For horizontal wires we use the same convention but rotated 90 degrees counterclockwise, i.e. if the 1s are on the west half of the non-red bits, we carry a logical 0-bit.

The two crucial aspects of a gadget are its *charging behavior* and the *relation* it defines. The charging behavior is denoted by $X \vdash Y$ where X and Y are subsets of $\{E, N, S, W\}$, written as a subword of $ENWS$ for brevity, and the relation is denoted by $X \in \{Y_1, Y_2, \dots, Y_k\}$ where again X is again subword of $ENWS$, and Y_i are subsets of $\{0, 1\}^{|X|}$.

The interpretation of the charging behavior is that if the wires on sides indicated by X are all charged, then the wires on sides Y are charged as well. We may also write $X_1, X_2, \dots, X_n \vdash Y$ to mean that each $X_i \vdash Y$ holds individually. The interpretation of the relation is that if the wires on all sides are charged, then their bit values (interpreted as in the last paragraph) must belong to the stated set. The charging behavior and relation of a gadget together constitute its *specs*.

Example 1: Consider a hypothetical gadget with the specs

$$W \vdash NE; WNE \in \{001, 010, 100\}.$$

This means that the gadget has wires on the west, north and east sides. If the west wire is charged, meaning that a preimage contains a signal in an operating phase, then the two others will be charged as well (in that preimage). In any such preimage, exactly one of the wires is in phase 1, and all such combinations can be realized in preimages where the rest of the boundary consists of 0s. \circ

We note that the specs refer to a particular orientation of the gate. When a gate is rotated 90 degrees counterclockwise, of course the permutation $(E\ N\ W\ S)$ is applied to all symbols appearing in the specification. In addition, when a wire turns counterclockwise from west to south or east to north, one must flip the bit on that wire in every tuple in the relation. Thus, in the relation, $ENWS = abcd$ turns into $ENWS = d\bar{a}b\bar{c}$, where $\bar{0} = 1, \bar{1} = 0$.

This convention for the meaning of 0 and 1 in a wire makes it easy to reason about large circuits, as we need not keep track of an orientation for individual wires. However, when reasoning about individual gadgets (and especially when rotating them) an easier point of view is that of orientations. A horizontal wire coming into a gadget from the west (resp. east) boundary is said to have *near* charge if its charge carries the bit 1 (resp. 0). Similarly, a vertical wire coming into a gadget from the north (resp. south) boundary is said to have *near* charge if its charge carries the bit 1 (resp. 0). Intuitively, looking at the two operating phases of the signal between any two red cells, the one closer to the center of the gadget is the near charge. The other operating phase is called a *far* charge. This determines the *affinity* of the charge.

If gadgets are thought of as manipulating the affinity of charges, the specs stay the same when we rotate them. Of course the convention between the conventions is straightforward. For the reader's comfort we include in all specs the affinity version of the relation, as a subset of $\{N, F\}^k$ for a suitable power k , with N and F denoting respectively the near and far charges.

4.1 Charging a wire: the charger-splitter combo

A typical preimage of a wire is messy, and does not propagate deterministically, so to get computation going the first thing we need to do is charge our wires. In other words, we want a gadget such that in every preimage, the wire on the boundary is charged, and there is a preimage where the wire is in this charged state and everything else on the boundary is zero. The *charger gadget*, shown in Figure 3, has such behavior.

In words, the precise property of the gadget is the following. The 4×2 rectangle at the top boundary, whose bottom half is inside the domain and which has the two black top cells of the wire in its middle (highlighted in Figure 3), is forced to contain the pattern W_0 or W_1 . For each of the two, the gadget has a preimage where the boundary is otherwise empty. Intuitively, the behavior of the charger gadget is quite simple: it charges the top wire in an arbitrary operating phase.

A charged wire that starts from a charger is somewhat useless on its own. In order to transmit information, we need charged wires with two ends. To achieve this, we combine the charger with the *splitter*, depicted in Figure 4, which splits

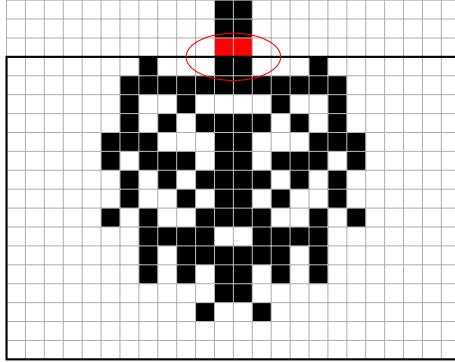


Figure 3: The charger gadget where one output is near and one is far. Specs: $\vdash N; N \in \{0, 1\}/\{\text{N}, \text{F}\}$. The gadget is inside the black rectangle, and we show the beginning of a wire on top.

a signal in two directions. In fact, we will only use the charger in combination with a splitter.

Note that there is a wire going straight through the splitter, so the north-south behavior is clear, and the wire on the east copies this information. In the shown orientation, the gadget has the side effect that the signal moving to the east is inverted. In terms of affinity, we copy affinity from the side where a bar of four black cells – the *handle bar* – sticks out of the wire (on top in the shown orientation).

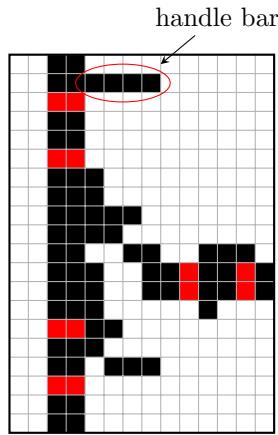


Figure 4: The splitter. Specs: $N, S \vdash ENS; ENS \in \{100, 011\}/\{\text{FFN}, \text{NNF}\}$.

Combining the charger and splitter, we obtain the *charged turn* gadgets shown in Figure 5. The intuitive behavior of these gadgets is that the input and output wires are charged and their signals are synchronized. Depending on the orientation, the signal may be flipped. In terms of affinity, if the signal meets the handle bar, affinity is kept.

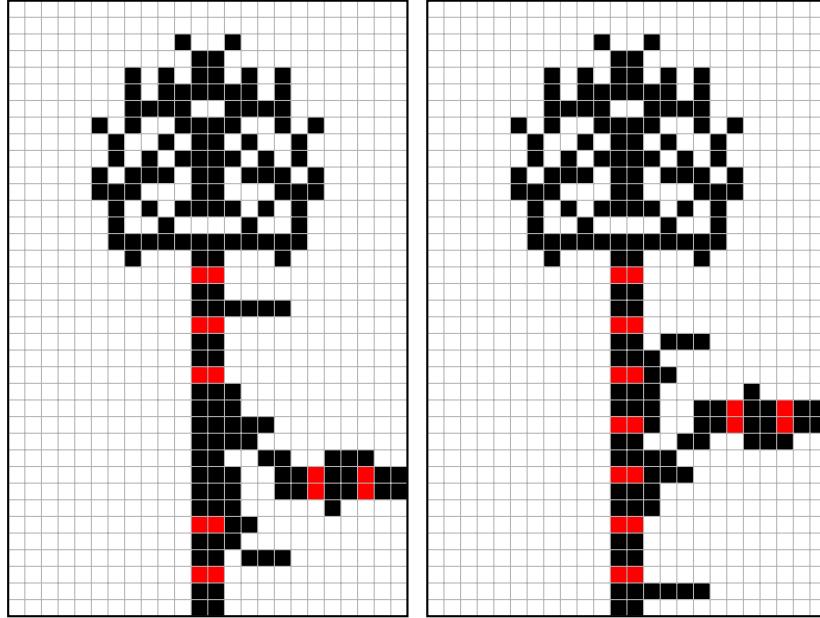


Figure 5: The charged turn gadgets. Specs: $\vdash ES$ for both, $ES \in \{01, 10\}/\{\text{NF}, \text{FN}\}$ for the first, $ES \in \{00, 11\}/\{\text{NN}, \text{FF}\}$ for the second.

4.2 Shifting and inverting a signal

The charged turn allows us to construct charged wires and turn them. However, for now the wires are cumbersome, if not impossible, to work with, since the phase of the wire stays in lockstep with the grid, and our gadgets do not align nicely modulo three. Thus, we introduce a gadget that allows free manipulation of the phase, namely the *inverter*, shown in Figure 6.

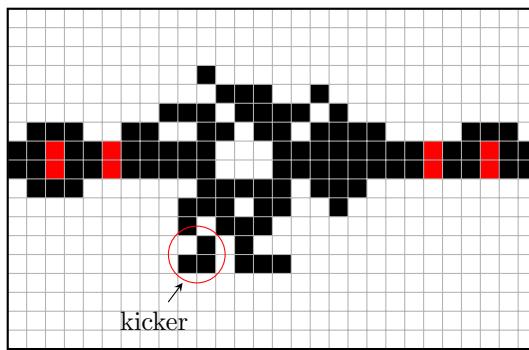


Figure 6: The inverter. Specs: $E \vdash W; EW \in \{01, 10\}/\{\text{NN}, \text{FF}\}$.

As a mnemonic, the reader may find it useful to visualize the contents of the bottom two inhabited rows as the “feet” of the gadget. In terms of these feet, the inverter kicks the signal with its smaller foot, or its *kicker* (in the figure,

the westmost one) to invert it. As a side effect it increments the phase of the signal by one, i.e. the horizontal distance between a red cells on the west side and one on the east side is $-1 \bmod 3$. The inverter gate propagates a charge in the direction of the kick, which in the figure is from east to west.

Since 2 and 3 are coprime, the inverter on its own can be used to invert and phase shift the signal. Namely we can first use 0 or 1 inverters to change the bit carried by a wire, and then two copies of it change the phase by $1 \bmod 3$ or four copies to change it by $-1 \bmod 3$. It seems likely that gadgets of similar size exist for other phase-signal transformations, but have not been able to find any.

4.3 Wire crossing

A *wire crossing* gadget is given in Figure 7. Note that the gadget performs no charging, and that as a side-effect, the phases are inverted (i.e. affinity is copied).

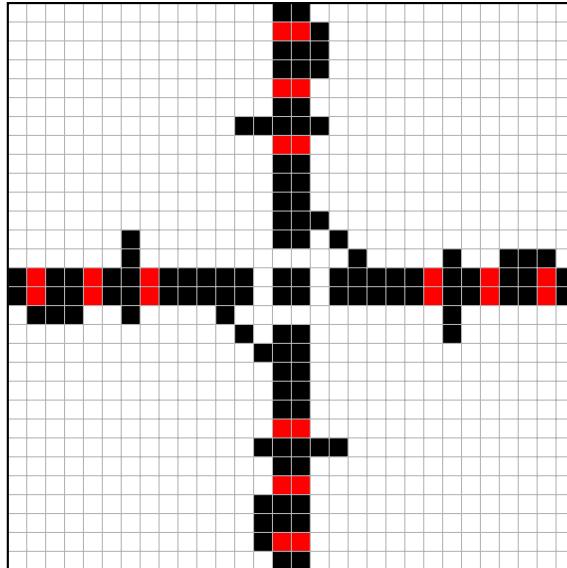


Figure 7: The wire crossing gadget. Specs: no charging, $ENWS \in \{0011, 0110, 1001, 1100\} / \{\text{NFNF}, \text{NNNN}, \text{FFFF}, \text{FNFN}\}$.

4.4 Any old logical gate

Now that we can freely connect wires and invert their values, all we need is a gate that, together with NOT, is universal. One such gate is specified in Figure 8. In terms of affinity, the top output is far if and only if both inputs are far. In global terms, in the shown orientation, seeing the west and east signal phases as the input, and north as the output, this is the $\neg(\neg W \wedge E)$ gate. By composing with inverters, one can easily produce any standard gate such as an AND gate or an OR gate in any orientation.

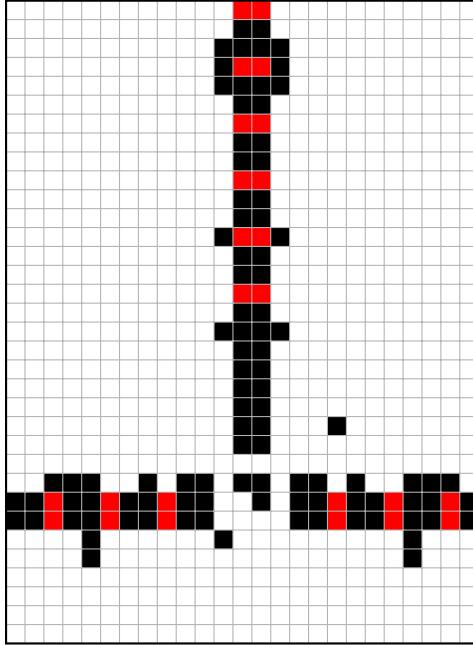


Figure 8: A gadget that performs some logical operation. Specs: no charging, $ENW \in \{111, 011, 100, 010\}/\{FNF, NNN, FFF, NNF\}$.

4.5 Constant charger

The above set of gadgets is already universal, but we nevertheless present also a gadget that introduces a fixed charge on a wire. It is called the *enforcer* and is depicted in Figure 9. As the enforcer sends a fixed charge, it does not determine two phases, so we can arbitrarily choose whether the signal it sends is 0 or 1 by aligning it appropriately. We choose to interpret the signal as being sent in the near phase. In the figure we use the usual red color to indicate the cells where the preimage always contains zero, i.e. the unused phase, and green denotes the other unused phase coming from the fact the F bit is never emitted.

5 Composite gadgets and abstract circuits

Our composite gadgets will correspond to the following set of *gate tiles*:

$$T = \{\square, \boxminus, \boxplus, \boxleftarrow, \boxrightarrow, \boxdot, \boxtimes, \boxwedge, \boxvee, \boxoplus, \boxotimes\}.$$

Before describing the gadgets, we give these symbols abstract component semantics, which mostly correspond to their usual meanings as used in digital logic. A *well-formed circuit* is any element of the SFT $X_T \subset T^{\mathbb{Z}^2}$ where in adjacent tiles all wires that reach the boundary of the tile must continue in the neighboring tile. If T is interpreted as a set of Wang tiles over the color set $C = \{\text{no wire}, \text{wire}\}$ (ignoring the fact that some combinations of colors correspond to several distinct tiles), then X_T is precisely the SFT defined by them as Wang tiles.

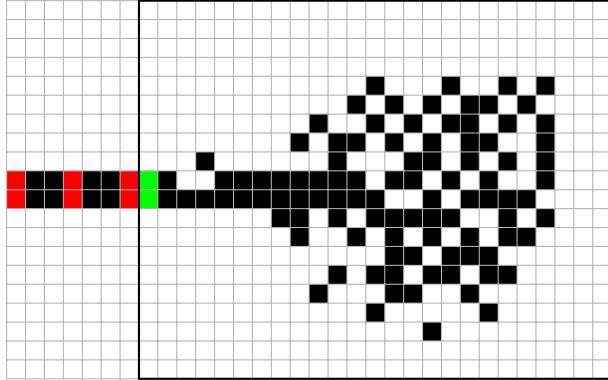


Figure 9: The enforcer. Specs: $\vdash W; W \in \{1\}/\{\text{N}\}$.

A circuit is *satisfiable* if it is possible to pick a *signal* $s \in \{0, 1\}$ for each of the wires in each tile reaching its boundary, so that the following hold:

- In the tiles $\square, \square, \square, \square, \square, \square$ the signals at both wires are the same.
- In \square , the signals in the two wires are distinct.
- In \square , the signal in the unique wire is 1.
- In \square , the signal in all the wires is the same.
- In \square , the north and south signals are equal, and the east and west signals are equal.
- In \square , the east signal is 1 if and only if at least one of the north and south signals is 1. (This is the logical OR operation.)

Write $Z_T \subset X_T \times ((\{0, 1, \perp\})^4)^{\mathbb{Z}^2}$ for the subshift encoding the satisfying assignments (where the $(\{0, 1, \perp\})^4$ -element in each cell codes the values of signals, and \perp is used iff there is no wire), and $f : Z_T \rightarrow T^{\mathbb{Z}^2}$ the natural projection. (For technical reasons, which will become clear later, the codomain is to be taken to be a full shift.) The image $f(Z_T) = Y_T \subset X_T$ is a sofic shift. The *circuit system* is the triple $(f, Z_T, T^{\mathbb{Z}^2})$.

Note that we are “missing” some obvious gates, like an AND gate and various orientations of gates, which would be useful for many engineering purposes, but they can be “simulated” easily (even in a precise sense as discussed in Section 7).

We now explain what our composite gadgets should do, and the correspondence with gate tiles. To each gate tile, we will associate a 450×450 pattern where there are four special positions on the east, north, west and south boundaries. A wire enters the gadget from a special position if and only if the corresponding gate tile has an outgoing wire on that side. Thus the SFT rule of X_T translates into “wires should continue across the borders of adjacent composite gadgets”.

The rest of the boundary of each composite gadget is all zero. In every preimage of a composite gadget, all the boundary wires are charged in one of two phases. The two phases used are the ones where the 1s are on either

side of the boundary (the phase where we have zeroes around the boundary never extends to a preimage of a gadget). We refer to the choice of which two phases are used for bits as the *phase alignment*, and the aforementioned choice of positioning of the used phases is referred to as the *neutral alignment*.

Interpreting the signal values as in the previous section, i.e. a representing a 1-signal by having the 1s on the east and south sides of the boundary, the allowed combinations for the signals should be precisely the ones described above for the abstract semantics, and furthermore for any such choice the rest of the boundary of each composite gadget should be fillable with zeroes.

We construct such gadgets in two parts. First, we give 90×90 blocks that simply connect the corresponding gadget from Section 4 to the same position on the border (we position our wire boundaries at the 29th and 30th column/row), in some cases using some inverters. We cannot simply put the resulting 90×90 patterns together, since

- the wires are not necessarily charged, and
- the phase alignments of neighboring patterns do not match.

To solve these issues, we introduce horizontal (resp. vertical) “charged wire gadgets” of shape 180×90 (resp. 90×180). These wire gadgets should charge the wires, and we construct one for each of the nine possible alignment changes. We can then simply position our basic gadgets in the middle 90×90 squares of 450×450 squares, and use the remainder of the space to put charged wire gadgets on the appropriate sides, changing all phase alignments to the neutral one.

We provide Golly-compatible patterns corresponding to this description in the repository [19]. If one has memorized the specs of the basic gadgets given in Section 4, it is a reasonably simple eyeballing exercise to verify that they indeed correctly implement the semantics of the gate tiles. In addition to having performed this eyeballing exercise several times ourselves, we have checked the semantics of the composite gadgets (but not the 450×450 squares) by a SAT solver. A Python script that performs this check for all the gadgets of Section 4, the corresponding 90×90 squares, and the 180×90 charged wires, can also be found in [19].

Once we have the gadgets, it is easy to see that for any satisfiable circuit, the corresponding tiling by composite gadgets has a preimage: satisfy the abstract circuit, copy the signal values (using the interpretation of values which is consistent among neighboring composite gadgets), and fill the rest of the boundaries with zeroes. On the other hand, if the composite gadget has a preimage, then already the values on the wires prove the satisfiability of the circuit. Thus, we obtain the following theorem.

Theorem 1. *There exists a substitution $\tau : T \rightarrow \{0, 1\}^{450 \times 450}$ such that for $x \in X_T$, $\tau(x)$ has a g-preimage if and only if $x \in Y_T$.*

6 A concrete example: the Jeandel-Rao tile set

Before delving deeper into universality, we show how to simulate an arbitrary Wang tile set with at most four edge colors in a direct fashion, without explicitly referring to universality. We also optimize the size of the simulation slightly,

from 450×450 to 270×270 , by not resetting to neutral phase alignment between patterns, but simply directly using, between two composite gadgets, the correct charged wire gadget connecting their phase alignments.

The generalization of the construction of this section to an arbitrary tile set is straightforward, and can be used to give an alternative proof of Lemma 3, which for small tile sets gives smaller and simpler implementations.

Represent the colors of a Wang tile by eight bits x_1, x_2, \dots, x_8 , with the convention that (x_2, x_1) are the west color, (x_3, x_4) the north color, (x_5, x_6) is the east color, and (x_8, x_7) is the south color. A set of k Wang tiles then becomes a set of k words in $\{0, 1\}^8$, and we can represent it as a logical formula in disjunctive normal form with k clauses.

Consider Figure 10. Repeat the rows 17–28 (0-indexed) $k - 2$ times. More precisely, seeing the figure as a matrix, add $12(k - 2)$ more rows immediately after the 28th, copying the contents periodically from rows 17–28, and leave the first 17 and last 15 rows intact.

Next, observe that, picking the ?s to be either horizontal wires or NOT-gates, we can interpret the binary tree of 7 OR-gates followed by a NOT gate as a conjunctive clause (using De Morgan’s law, i.e. inverting the inputs). We can then interpret the column of ORs on the right as a disjunction of clauses.

Finally, we replace the gate tiles by the 90-by-90 blocks from the appendix, and connect them with charged wire gadgets – the minor additional complication is now that we need to pick the phases context-sensitively, i.e. we have to make sure that the outgoing phase of a wire going out from one gadget is matched with the incoming phase of the neighboring one. This can be done, as the appendix lists the gadgets for connecting any pair of phases.

Theorem 2. *There exists a totally periodic configuration $x \in \{0, 1\}^{\mathbb{Z}^2}$ with periods 6210×37800 that has a g-preimage, but no periodic preimage.*

Proof. The aperiodic Jeandel-Rao tile set of [10] uses only four colors on each side and has 11 tiles. With $k = 11$, after the rows 19–31 have been repeated $k - 2$ times we have an array with $17 + 12 \cdot 9 + 15 = 140$ rows and 23 columns. Replacing each cell by the appropriate 270×270 pattern, we get the claimed periods of 6210×37800 . \square

The fundamental domain of the configuration x is shown in Figure 1. We have not seriously tried to optimize the periods, apart from seeking to produce a somewhat readable figure.

Question 3. *What are the smallest periods (for example in terms of the size of the fundamental domain) of a periodic configuration that has a g-preimage, but no periodic preimage?*

7 Universality

In this section, we give more precise universality definitions, and state a stronger version of Theorem 1. Our precise definition of simulation is a bit involved, and readers without a background or an interest in universality per se may be better off skipping straight to Section 8, and using the proof of Theorem 1 and the construction from Section 6 to replace the universality discussion, as all of the proofs are quite easy.

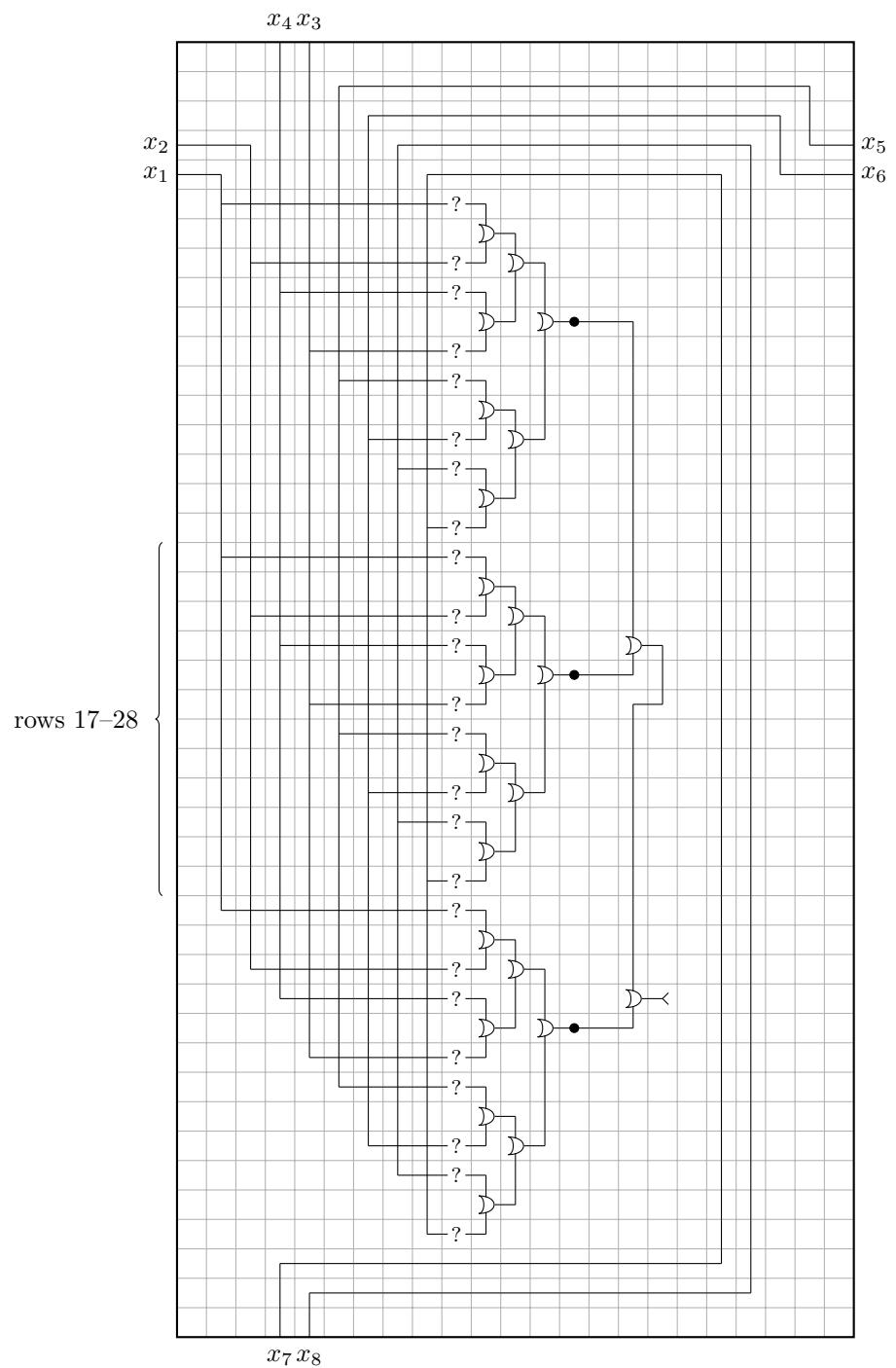


Figure 10: The “blueprint” for simulating a tile set by preimages of g .

Definition 1. Let $X \subset A^{\mathbb{Z}^2}$ be a subshift. For $m, n \in \mathbb{N}$, the (m, n) -blocking of X is the \mathbb{Z}^2 -dynamical system $(X, m\mathbb{Z} \times n\mathbb{Z})$, where \mathbb{Z}^2 acts as $(i, j) \cdot x = \sigma^{(mi, nj)}(x)$. It can be seen as a subshift over the alphabet of blocks $A^{m \times n}$ in a natural way. Every block map $\psi : X \rightarrow Y$ lifts into a block map between blockings $\psi^{m \times n} : (X, m\mathbb{Z} \times n\mathbb{Z}) \rightarrow (Y, m\mathbb{Z} \times n\mathbb{Z})$, called the (m, n) -blocking of ψ . We denote it simply by ψ when there is no danger of confusion.

We interpret the (m, n) -blocking of a subshift X as a set of configurations with a superimposed $m \times n$ grid. A block map $f : (X, m\mathbb{Z} \times n\mathbb{Z}) \rightarrow Y$ needs to satisfy the relation $f(\sigma^{(mi, nj)}(x)) = \sigma^{(i, j)}(f(x))$ for all $(i, j) \in \mathbb{Z}^2$ and $x \in X$, that is, a shift by $(1, 0)$ on the codomain side corresponds to a shift by $(m, 0)$ on the domain side, and similarly for $(0, 1)$ and $(0, n)$. Similarly, a substitution $\tau : A \rightarrow B^{m \times n}$ of shape $m \times n$ can be interpreted as a block map $\tau : A^{\mathbb{Z}^2} \rightarrow (B^{\mathbb{Z}^2}, m\mathbb{Z} \times n\mathbb{Z})$.

In the definition of simulations, we discuss commutation of diagrams of *partial functions*. To make sense of this, we make explicit our formalism of partial functions.

Definition 2. A partial function $f : A \nrightarrow B$ consists of a domain $\text{dom}(f) \subset A$ and an underlying function $\text{fun}(f) : \text{dom}(f) \rightarrow B$, which we may abusively denote by f . Two partial functions are equal if they have equal domains and underlying functions. We say f is injective or surjective if $\text{fun}(f)$ is, and bijective if $\text{dom}(f) = A$ and $\text{fun}(f)$ is bijective. The composition $g \circ f : A \nrightarrow C$ of two partial functions $f : A \nrightarrow B$ and $g : B \nrightarrow C$ is the partial function with domain $\text{dom}(f) \cap \text{fun}(f)^{-1}(\text{dom}(g))$ and the obvious underlying function.

Definition 3. Let $\psi : X \rightarrow B^{\mathbb{Z}^2}$ be a block map, where $X \subset A^{\mathbb{Z}^2}$, and let $C \subset B$. The alphabet co-restriction to C of ψ is the partial block map $\psi_C : X \nrightarrow B^{\mathbb{Z}^2}$ with domain $\psi^{-1}(C^{\mathbb{Z}^2})$ and underlying function the restriction of ψ .

Definition 4. Let $\psi : X \rightarrow B^{\mathbb{Z}^2}, \phi : Y \rightarrow D^{\mathbb{Z}^2}$ be block maps, where $X \subset A^{\mathbb{Z}^2}, Y \subset C^{\mathbb{Z}^2}$ are subshifts of finite type. We say ψ weakly simulates ϕ if there exist $m, n \in \mathbb{N}$, an injective substitution $\tau : D \rightarrow B^{m \times n}$, and a surjective partial block map $h : (X, m\mathbb{Z} \times n\mathbb{Z}) \nrightarrow Y$, such that the following diagram of partial functions commutes:

$$\begin{array}{ccc} (X, m\mathbb{Z} \times n\mathbb{Z}) & \xrightarrow{h} & Y \\ \downarrow \psi_{\tau(D)} & & \downarrow \phi \\ (B^{\mathbb{Z}^2}, m\mathbb{Z} \times n\mathbb{Z}) & \xleftarrow{\tau} & D^{\mathbb{Z}^2} \end{array} \quad (2)$$

We say ψ semiweakly simulates ϕ if h admits a block map section, that is, there exists a block map $\chi : Y \rightarrow (X, m\mathbb{Z} \times n\mathbb{Z})$ with $h \circ \chi = \text{id}_Y$. We say ψ strongly simulates ϕ if h is bijective (and hence a conjugacy).

Note that the leftmost arrow in the above diagram is an alphabet co-restriction of a blocking of ψ . In particular, the domain of h equals $\phi^{-1}(\tau(D^{\mathbb{Z}^2}))$. The images $\tau(d) \in B^{m \times n}$ for $d \in D$ are sometimes called ‘‘macrotiles’’.

A weak simulation of ϕ by ψ can be defined by the data (m, n, τ, h) . In applications, we often want to produce such data algorithmically. It is important to use the convention that to specify the partial block map h , it suffices to specify

any block map $h' : (A^{\mathbb{Z}^2}, m\mathbb{Z} \times n\mathbb{Z}) \rightarrow C^{\mathbb{Z}^2}$ that agrees with h on $\text{dom}(h)$, and otherwise may produce configurations that are not even in Y . In particular, we do not need to actually compute $\text{dom}(h)$.

The full shift on the image side should be seen as an “ambient full shift” around the sofic image that we are actually interested in. On the preimage side it is natural to use SFTs rather than full shifts, because alphabet co-restrictions of $\psi : X \rightarrow B^{\mathbb{Z}^2}$ can turn the domain into a proper SFT even if X is a full shift.

We first prove that the simulation relations respect conjugacies and are transitive.

Lemma 1. *Let $X \subset A^{\mathbb{Z}^2}$ be an SFT and $\psi : X \rightarrow B^{\mathbb{Z}^2}$ a block map. Let $\alpha : X \rightarrow Y$ be a conjugacy. Then ψ strongly simulates $\psi \circ \alpha^{-1} : Y \rightarrow B^{\mathbb{Z}^2}$, and the simulation can be effectively computed from the local rules of ψ and α .*

Proof. We can choose $m = n = 1$, $\tau = \text{id}_B$ and $h = \alpha$ in the definition of the strong simulation. \square

Lemma 2. *Weak (resp. semiweak, strong) simulation is transitive, and the composition of two simulations can be effectively computed from its components.*

Proof. Suppose $X \subset A^{\mathbb{Z}^2}, Y \subset C^{\mathbb{Z}^2}, Z \subset E^{\mathbb{Z}^2}$. Suppose $\psi : X \rightarrow B^{\mathbb{Z}^2}$ weakly simulates $\phi : Y \rightarrow D^{\mathbb{Z}^2}$ with data (m, n, τ, h) and $\phi : Y \rightarrow D^{\mathbb{Z}^2}$ weakly simulates $\chi : Z \rightarrow F^{\mathbb{Z}^2}$ with data (m', n', τ', h') . We claim that ψ weakly simulates χ with data $(mm', nn', \tau \circ \tau', h' \circ h)$ where $\tau \circ \tau' : F \rightarrow B^{mm' \times nn'}$ is a composition of substitutions, and $h' \circ h : (X, mm'\mathbb{Z} \times nn'\mathbb{Z}) \rightarrow Z$ is a composition of partial functions.

Consider the following diagram of partial functions:

$$\begin{array}{ccccc} (X, mm'\mathbb{Z} \times nn'\mathbb{Z}) & \xrightarrow{h} & (Y, m'\mathbb{Z} \times n'\mathbb{Z}) & \xrightarrow{h'} & Z \\ \psi_{\tau(\tau'(F))} \downarrow & & \phi_{\tau'(F)} \downarrow & & \chi \downarrow \\ (B^{\mathbb{Z}^2}, mm'\mathbb{Z} \times nn'\mathbb{Z}) & \xleftarrow{\tau} & (D^{\mathbb{Z}^2}, m'\mathbb{Z} \times n'\mathbb{Z}) & \xleftarrow{\tau'} & F^{\mathbb{Z}^2} \end{array} \quad (3)$$

The right half is the diagram for the simulation of χ by ϕ , which commutes by assumption. Thus, it suffices to prove that the left half commutes.

If we apply the (m', n') -blocking operation to the diagram for the simulation of ϕ by ψ , we obtain the following commutative diagram:

$$\begin{array}{ccc} (X, mm'\mathbb{Z} \times nn'\mathbb{Z}) & \xrightarrow{h} & (Y, m'\mathbb{Z} \times n'\mathbb{Z}) \\ \psi_{\tau(D)} \downarrow & & \phi \downarrow \\ (B^{\mathbb{Z}^2}, mm'\mathbb{Z} \times nn'\mathbb{Z}) & \xleftarrow{\tau} & (D^{\mathbb{Z}^2}, m'\mathbb{Z} \times n'\mathbb{Z}) \end{array} \quad (4)$$

The only difference with this diagram and the left half of (3) is the further subalphabet restriction on both vertical arrows, so it suffices to verify that the domains of $\phi_{\tau(\tau'(F))}$ and $\tau \circ \phi_{\tau'(F)} \circ h$ are equal.

Suppose $x \in X$ is in the domain of $\psi_{\tau(\tau'(F))}$. Since $\tau(\tau'(F)) \subset \tau(D)^{m' \times n'}$, in particular x is in the domain of $\psi_{\tau(D)}$, which equals the domain of $\tau \circ \phi \circ h$ by the commutativity of (4). We also have $\tau \phi h(x) = \psi(x) \in \tau(\tau'(F)^{\mathbb{Z}^2})$. Because τ is injective, this implies $\phi h(x) \in \tau'(F)^{\mathbb{Z}^2}$, and hence x is in the domain of $\tau \circ \phi_{\tau'(F)} \circ h$.

Conversely, if x is in the domain of $\tau \circ \phi_{\tau'(F)} \circ h$, then it is in the domain of $\tau \circ \phi \circ h = \psi_{\tau(D)}$ and $\psi(x) = \tau \phi h(x) \in \tau(\tau'(F^{\mathbb{Z}^2}))$.

For semiweak and strong universality we simply note that the properties of having a block map section and being a conjugacy are preserved in composition. \square

Strong simulation is the “best” we can hope for, in a sense. Semiweak simulation improves on weak simulation – due to the factor map we inherit all complexity from ψ (things like periodicity and asymptotic Kolmogorov complexity), and due to the section *some* subset of the fibers contains no extra information.

Definition 5. *If a block map $\psi : X \rightarrow B^{\mathbb{Z}^2}$ weakly (resp. strongly, semiweakly) simulates every block map $\phi : Y \rightarrow D^{\mathbb{Z}^2}$, where X and Y are SFTs, then we say that the triple $(\psi, X, B^{\mathbb{Z}^2})$ is weakly (resp. strongly, semiweakly) universal. We say it is effectively weakly (resp. strongly) universal if we can additionally compute a simulation (m, n, τ, h) from the local rule of ϕ . For effective semiweak universality, we require that a section can also be computed.*

Theorem 3. *The circuit system $(f, Z_T, T^{\mathbb{Z}^2})$ is effectively strongly universal.*

Proof. Consider any $\phi : Y \rightarrow D^{\mathbb{Z}^2}$. By Lemmas 1 and 2, we may apply a recoding to Y so that ϕ becomes a symbol map and Y becomes the SFT of tilings by a set of Wang tiles (in particular the side colors determine the tile). In particular, we assume $Y \subset C^{\mathbb{Z}^2}$ where C is a set of Wang tiles.

We now construct macrotiles that simulate the symbols in C . Pick large (but otherwise arbitrary) m, n and for each $c \in C$, in the corresponding macrotile $\tau(c) \in T^{m \times n}$ we draw wires from each border and connect them to the bottom of a computation zone in the middle of the macrotile (this part is independent of c). The wires therefore synchronize a sequence of bits between adjacent macrotiles, and the sequence on each side represents the color on the corresponding side of one of the Wang tiles in C . In the computation zone of the macrotile corresponding to c , we simply check that the bit sequences are the colors of a valid Wang tile that maps to c under ϕ : it is clear that we can draw any classical digital circuit with our tiles, and such circuits are computationally universal.

The map h simply maps each macrotile to the Wang tile in C coded by the colors. If we perform computations in the computation zone deterministically, h will be a bijection, so this is indeed a strong simulation. \square

Lemma 3. *If a block map $\psi : C^{\mathbb{Z}^2} \rightarrow D^{\mathbb{Z}^2}$ is weakly (resp. strongly, semiweakly) universal, then it is effectively so.*

Proof. We observe that ψ weakly (resp. strongly, semiweakly) simulates the circuit system $(f, Z_T, T^{\mathbb{Z}^2})$ with a fixed simulation (m, n, τ, h) . Given any $\phi : Y \rightarrow D^{\mathbb{Z}^2}$, since $(f, Z_T, T^{\mathbb{Z}^2})$ is effectively strongly universal by Theorem 3, it simulates ϕ by some simulation (m', n', τ', h') that we can compute from a description of ϕ and Y . By Lemma 2, $(mm', nn', \tau \circ \tau', h' \circ h)$ is a simulation of ϕ by ψ that can be computed effectively from (m, n, τ, h) and (m', n', τ', h') .

The simulation of ϕ by $(f, Z_T, T^{\mathbb{Z}^2})$ is strong (thus it is weak and semiweak), proving the weak and strong cases. For the semiweak case, we observe that since h' is bijective, its inverse is a section, and inverting an invertible block map can

be done effectively. By assumption we can also compute a section χ for h . Then $\chi \circ (h')^{-1}$ is a section for $h' \circ h$. \square

Theorem 4. *Game of Life is semiweakly universal as a block map.*

Proof. The proof of Theorem 1 actually shows that Game of Life semiweakly simulates the circuit system $(f, Z_T, T^{\mathbb{Z}^2})$. Namely we showed weak universality, by constructing the macrocells $\tau(T)$ and explained the map h (which interprets the bits on our Game of Life wires as bits on the abstract wires carried in Z_T on top of the gate tiles).

For semiweak universality we need a section for h . The macrocells corresponding to gate tiles were constructed so that we can pick zero borders away from the wires. For each gate tile t and each tuple of bits (b_1, b_2, b_3, b_4) carried on the wires, we pick for the macrotile corresponding to t an arbitrary (but always the same) preimage for the macrotile where the Game of Life wires carry signals corresponding to the b_i , and the section of h simply performs the corresponding substitution. \square

Question 4. *Is Game of Life strongly universal as a block map?*

8 Corollaries

Here we list the theorems mentioned in the introduction, and explain how they are obtained.

Theorem 5. *Given a finite-support configuration, it is NP-complete whether it admits a g -preimage.*

Proof. This problem is in NP by [18, Theorem 1], so it suffices to prove NP-hardness.

Consider the circuit system $f : Z_T \rightarrow T^{\mathbb{Z}^2}$, and as usual let X_T be the set of well-formed circuits and $Y_T = f(Z_T)$. From a given SAT instance S we can easily compute a polynomial-size finite-support X_T -configuration x_S which is in Y_T if and only if the instance is satisfiable. Readers that skipped Section 7 will note that the result immediately follows from the statement of Theorem 1.

To readers that did not skip Section 7, we explain how to deduce the result from semiweak universality by diagram-chasing. By Theorem 4, there is a semiweak simulation (m, n, τ, h) of the circuit system by g (indeed this is the simulation from Theorem 1).

We claim that $\tau(x_S)$ is in the image of g if and only if the SAT instance S is satisfiable. This is immediate from the simulation diagram (2) in this context:

$$\begin{array}{ccc} (\{0, 1\}^{\mathbb{Z}^2}, m\mathbb{Z} \times n\mathbb{Z}) & \xrightarrow{h} & Z_T \\ g_{\tau(T)} \downarrow & & \downarrow f \\ (\{0, 1\}^{\mathbb{Z}^2}, m\mathbb{Z} \times n\mathbb{Z}) & \xleftarrow{\tau} & T^{\mathbb{Z}^2} \end{array}$$

Namely, if S is satisfiable, then $x_S \in T^{\mathbb{Z}^2}$ models a satisfiable circuit and has an f -preimage $z \in Z_T$. By the assumption that h is surjective, we find an h -preimage $y \in \{0, 1\}^{\mathbb{Z}^2}$ for z such that $g(y) = \tau(x_S)$. In particular, $\tau(x_S)$ has a g -preimage.

Conversely, if $\tau(x_S)$ has a g -preimage $y \in \{0, 1\}^{\mathbb{Z}^2}$, then by commutation we have $f(h(y)) = x_S$, showing S is satisfiable. \square

Theorem 6. *The set of orphans for g is coNP-complete.*

Proof. This is trivially in NP. NP-hardness follows from Theorem 5 and [18, Theorem 3], which states that if a rectangular pattern is padded with a thickness-4 border of dead cells and this padded pattern is not an orphan for g , then the extension to an infinite configuration by dead cells admits a g -preimage. Alternatively, it can be proved as a finitary version of Theorem 5. \square

Theorem 7. *Given a subshift of finite type Y , we can effectively compute $m, n \in \mathbb{N}$ and an $m \times n$ -periodic configuration $x \in \{0, 1\}^{\mathbb{Z}^2}$ such that the system $(g^{-1}(x), m\mathbb{Z} \times n\mathbb{Z})$ admits Y as a split factor.*

Proof. Consider the block map $\phi : Y \rightarrow \{0\}^{\mathbb{Z}^2}$ from Y to a singleton subshift. By effective semiweak universality of g , we can compute a semiweak simulation (m, n, τ, h) of ϕ by g . The substitution image $x = \tau(0^{\mathbb{Z}^2})$ is $m \times n$ -periodic.

Consider now the diagram of the simulation:

$$\begin{array}{ccc} (\{0, 1\}^{\mathbb{Z}^2}, m\mathbb{Z} \times n\mathbb{Z}) & \xrightarrow{h} & Y \\ g_{\tau(0)} \downarrow & & \downarrow \phi \\ (\{0, 1\}^{\mathbb{Z}^2}, m\mathbb{Z} \times n\mathbb{Z}) & \xleftarrow{\tau} & \{0\}^{\mathbb{Z}^2} \end{array}$$

The domain of $g_{\tau(0)}$ is exactly $g^{-1}(x)$, and it also equals the domain of h . Since the simulation is semiweak, h admits a block map section $\chi : Y \rightarrow (g^{-1}(x), m\mathbb{Z} \times n\mathbb{Z})$, which we can also effectively compute. Hence Y is a split factor of $(g^{-1}(x), m\mathbb{Z} \times n\mathbb{Z})$, as required. \square

The following theorems are immediate corollaries.

Theorem 8. *It is undecidable (specifically, Π_1^0 -complete) whether a given totally periodic configuration $x \in \{0, 1\}^{\mathbb{Z}^2}$ has a g -preimage.*

Proof. Given an SFT Y , construct the configuration $x \in \{0, 1\}^{\mathbb{Z}^2}$ as in Theorem 7. Now x is a g -preimage if and only if Y is nonempty. The latter is undecidable by [2]. \square

Theorem 9. *Given a totally periodic configuration $x \in \{0, 1\}^{\mathbb{Z}^2}$ that has a preimage under g , it is undecidable whether it has a totally periodic g -preimage. Given a totally periodic x , it is Σ_1^0 -complete whether it has a totally periodic g -preimage.*

Proof. Given an SFT Y , construct the configuration $x \in \{0, 1\}^{\mathbb{Z}^2}$ as in Theorem 7. If Y is nonempty, then the corresponding periodic configuration x has a preimage. Furthermore, if Y has a periodic point, then by semiweak universality so does x (by applying the section). Conversely, any periodic preimage of x maps to a periodic point of Y . It follows that Y has a periodic point if and only if x has a periodic preimage. Given a nonempty SFT, it is undecidable whether it contains a totally periodic point [9, Theorem 5], proving the first claim.

As for the last sentence, [9, Theorem 5] indeed shows that the existence of a totally periodic point in a given SFT is Σ_1^0 -hard. It is in Σ_1^0 since we simply need to exhibit a totally periodic point to prove it. \square

We note that Theorem 9 does not follow directly from weak universality, since we cannot prove the existence of a totally periodic preimage without the section. More concretely, take any aperiodic SFT X and consider the block map $h : X \times \{0, 1\}^{\mathbb{Z}^2} \rightarrow \{0, 1\}^{\mathbb{Z}^2}$ defined by $h(x, y) = g(y)$. Then h is weakly universal, but no configuration has a totally periodic h -preimage, so the problems of Theorem 9 are trivially decidable.

Theorem 10. *There exists a totally periodic configuration $x \in \{0, 1\}^{\mathbb{Z}^2}$ that has a g -preimage but no computable g -preimage.*

Proof. There exists an SFT Y with no computable configurations [13]. Construct the configuration x as in Theorem 7, and observe that x cannot have a computable g -preimage, since its h -image would be a computable configuration in Y . \square

Theorem 11. *There exists a totally periodic configuration that has a g -preimage, but every preimage has $\Omega(n)$ Kolmogorov complexity in every $n \times n$ -pattern.*

Proof. The proof is similar to the ones above, but uses the existence of SFTs with maximal Kolmogorov complexity in all patterns from [3]. \square

Of course, the Kolmogorov complexity of the g -preimages is much lower than that of the factor – the hidden constant in the $\Omega(n)$ is very small.

9 Implementation and verification of SAT-based searches

We constructed the gadgets of Section 4 using two different SAT-based search programs and some manual work. Both programs are based on the following idea. Fix a finite set $D \subset \mathbb{Z}^2$ of cells and a set $\mathcal{F} \subset \{0, 1\}^D$ of possible “forced preimages”. Our goal is to produce a single finite pattern P such that the set $g^{-1}(P)|_D = \{Q|_D \mid Q \in g^{-1}(P)\}$ of shape- D subpatterns of its preimages is exactly \mathcal{F} . We see this as an optimization problem: there is a hard constraint $\mathcal{F} \subset g^{-1}(P)|_D$, and the goal is to minimize the cardinality of $g^{-1}(P)|_D \setminus \mathcal{F}$, hopefully to zero. We also need to impose additional hard constraints, such as P containing a wire at a certain position and 0-cells on the rest of its border, or relaxations, like only considering preimages that have a valid “incoming” signal on top of a wire of P .

The first algorithm is essentially a backtracking hill-climbing optimizer, and the second one is a genetic program. Both programs are written in Python and use the PySat library [8] to invoke the Glucose 4.1 SAT solver [1]. The source code for the hill climber is available at [20].

9.1 The hill climber

The hill climber program constructs the pattern P iteratively, starting from the empty pattern and adding one or more specified cells at a time. It expects the following parameters:

- A finite set $\{D_1, \dots, D_n\}$ of nonempty finite subsets of \mathbb{Z}^2 . Their union $D = \bigcup_{i=1}^n D_i$ is the domain of the forced preimages.

- A finite set $\{q_1, \dots, q_k\}$ of finite or infinite patterns, given as functions $q_i : \mathbb{Z}^2 \rightarrow \{0, 1, \perp\}$ with \perp meaning an unspecified cell.
- For each $i \in \{1, \dots, k\}$, a set $\mathcal{F}_i \subset \{0, 1\}^D$ of forced patterns. The semantics is that restricting to D those preimages that are compatible with q_i should produce exactly the set \mathcal{F}_i .
- A finite or infinite pattern, again given as a function $p : \mathbb{Z}^2 \rightarrow \{0, 1, \perp\}$. It represents a constraint on P : if $p(\vec{v}) = b \in \{0, 1\}$, then we must have $P_{\vec{v}} = b$.
- Several technical parameters that guide the search process.

The program proceeds in rounds. On each round, we enumerate the outer border ∂P of the current candidate pattern P , which consists of those cells that are not in the domain of P but have an immediate neighbor (orthogonal or diagonal) that is. For each $\vec{v} \in \partial P$ and $b \in \{0, 1\}$, we extend P into a new pattern P' by specifying $P'_{\vec{v}} = b$, and compute its *score* $s(P')$ as defined below. The extension with the lowest score will replace P . If no extension has a strictly lower score than P , then we enumerate all extensions by two adjacent cells on the border, then three, and so on up to some bound. If no better extension is still found, we repeatedly try to extend P by randomly specifying the values of a randomly chosen rectangle of cells near the border of P . After some number of tries, we give up and backtrack, choosing the next best extension of the previous round instead. The program keeps track of the entire history of the search, so it can backtrack as far as needed.

The score of a candidate pattern P is defined as

$$s(P) = \sum_{i=1}^k \sum_{j=1}^n \frac{1}{|D_j|} \log \left(1 + \sum_{R \in \mathcal{Q}_{(i,j)}} \frac{M(i, j, R)}{|D_j|} \right)$$

if it is *valid* (see below), and $s(P) = \infty$ if it is invalid. Here, $\mathcal{Q}_{(i,j)}$ is the set of patterns $R = Q|_{D_j}$ where $Q \in g^{-1}(P)$ is a preimage that agrees with q_i , minus those in $(\mathcal{F}_i)|_{D_j}$, and

$$M(i, j, R) = 1 + \max_{Q \in \mathcal{F}_i} |\{\vec{v} \in D_j \mid R_{\vec{v}} = Q_{\vec{v}}\}|.$$

is the largest number of cells on which R agrees with some pattern of \mathcal{F}_i . The idea is that we want to encourage the program to quickly get rid of preimages that closely resemble some forced preimage, as it should intuitively become more difficult as the pattern P grows.

Notice that since we have covered the domain D by the subsets D_1, \dots, D_k that are scored separately, having a score of 0 does not guarantee the property that restricting q_i -compatible preimages of P to D gives exactly \mathcal{F}_i , unless $k = 1$. The reason for the cover is efficiency: the size of each $\mathcal{Q}_{(i,j)}$ is initially exponential in that of D_j . However, as P grows, the subsets $\mathcal{Q}_{(i,j)}$ shrink, and when two of them become small enough, the program replaces the respective sets $D_{j(1)}$ and $D_{j(2)}$ with their union, thus decreasing k .

Let us now discuss the validity of patterns and the ways to verify it. There are three reasons a pattern P can be invalid. The first is that for some $\vec{v} \in \mathbb{Z}^2$ we have $p(\vec{v}) = b \in \{0, 1\}$ and $P_{\vec{v}} = 1 - b$. This can be verified immediately

after extending P by the cell \vec{v} . The second is that for some $i \in \{1, \dots, k\}$ and $Q \in \mathcal{F}_i$, there is no preimage $R \in g^{-1}(P)$ that is compatible with q_i and satisfies $R|_D = Q|_D$. This can be verified with a single call to a SAT-solver: it is simple to construct a SAT instance that is unsatisfiable if and only if the above property holds.

The third reason is more subtle. Consider the situation that for some $i \in \{1, \dots, n\}$ there are two preimages $Q, R \in g^{-1}(P)$ which are compatible with q_i , such that $Q|_D \in \mathcal{F}_i$, $R|_D \notin \mathcal{F}_i$, and $Q_{\vec{v}} = R_{\vec{v}}$ for every cell \vec{v} near the border of P . We call this pair of preimages a *diamond*. Then for every extension P' of P , either both Q and R or neither of them extend to q_i -compatible preimages of P' . Unless $Q|_D$ has another preimage that is not part of a diamond, the pattern $R|_D$ can never be forbidden from q_i -compatible preimages by extending P . We deem P to be invalid if it admits a diamond, as this potentially dangerous property can be checked with a bounded number of calls to a SAT-solver, whereas there does not seem to be an efficient way of determining whether each forced pattern admits a preimage that is not part of a diamond.

The final zero-score pattern P typically has an irregular shape, so it is completed into a rectangle one cell at a time, favoring 0-cells unless doing so would result in an invalid pattern. The constraint of admitting preimages with only 0-cells near the border is not enforced by the program, so it must be done manually. We added and removed 1-cells near wires in a trial-and-error fashion until the property held.

The program has various additional features that we have not fully described here. We only mention a forced preference to filling small crevices on the border of P before other cells, the possibility of forcing P to be periodic in some direction, and support for multithreading.

The hill climber program was used to find all patterns of Section 4 except the charger.² We note that the patterns were found with various early iterations of the program with different scoring functions. Together with the random component of the search, this means that the current program may produce vastly different gadgets.

9.2 The genetic algorithm

Like the hill-climbing algorithm, the genetic algorithm attempts to find a pattern that forces certain behavior for the preimage. Unlike the hill-climbing program, the genetic algorithm was not written as a general-purpose search program, but rather experimentally with hard-coded inputs and parameters. Since we only found the charger with this approach, we concentrate on the charger-specific optimization problem. As finding patterns with it requires extensive human effort, we have not included its source code.

What we want is to find a pattern $P \in \{0, 1\}^{[0, 2n-1] \times [0, m]}$ where

- the first two rows contain only a wire, which is precisely at the midpoint, i.e. $\text{supp}(P|_{[0, 2n-1] \times [0, 1]}) = [n-1, n] \times [0, 1]$;
- the remainder of the border of thickness 2 contains only zeroes;

²In fact, while we did not manage to directly charge a wire with the hill climber, we did find a “relaxer” using it. It has the specs $N \vdash E; EN \in \{00, 10\}$ meaning given a charged wire in a fixed phase, it can relax the phase to one of two choices. In combination with the enforcer this gives a (rather large) charger gadget.

- for at least two of the wire signals that can appear on $[n - 2, n + 1] \times [0, 1]$, they appear with zeroes on the remainder of the border;
- the number of different restrictions of preimages to $[n - 2, n + 1] \times [0, 1]$ is precisely two, namely we see precisely two different phases of the wire signal.

The first two items will hold at all times by our choices of initial patterns, and choice of how we modify them. We use a simple scoring function to check how far a pattern is from having the latter properties: If the third item fails, the pattern is immediately discarded (or given score $-\infty$). Then we simply count the number s of restrictions of preimages to $[n - 2, n + 1] \times [0, 2]$, and use the score $s - 2$; the maximal possible score is then 0, and it means that there are only two possibilities, which must be two wire signal phases, since the third item guarantees that two phases are even possible with zeroes elsewhere on the border of the preimage pattern. Both the check and the counting are easy to perform with a SAT solver.

We are then in the standard scheme of searching for a binary pattern under a scoring function, with a particular score indicating a pattern with the desired property. To solve this problem genetically, the algorithm keeps, at all times, track of a finite number n of patterns (we used an upper bound of 100). It performs a finite number of iterations on this population. In each iteration, it performs asexual and sexual genetic modifications to produce a number of new patterns. They are scored, and the best n are kept, with a minor twist: to keep the population from becoming too homogenous, we additionally compared “genetic similarity” (by simply counting the number of equal cells!), and did not include a pattern if it was too similar to multiple already included patterns.

Our reproduction rules were the following:

- asexual reproduction (i.e. cloning and mutation): we pick a pattern at random, then pick a small set of random small rectangles contained in its domain, and randomly pick the new contents of these rectangles.
- sexual reproduction: we pick two random patterns P_1, P_2 in our set, and construct a new pattern P by picking the i th row randomly from the corresponding row of P_1 or P_2 (this is known as *crossover*), for each $i \in [0, m]$. The choices of which pattern each row was taken from were not independent, but were generated by a Markov process that preferred to pick the row $i + 1$ from the same pattern that row i was copied from.

The idea of sexual reproduction was that some rows might somehow eliminate some preimages, and other rows might eliminate others. There were indeed situations during search where the algorithm was stuck for a while, after which sexual reproduction finally found a better solution. We do not know, however, whether this was truly a result of combining two patterns, or whether it could have been replaced by a more aggressive asexual reproduction rule.

9.3 Verification

The repository [19] contains a Python script that verifies the claimed properties of the gadgets of Section 4 (that they implement the claimed specs), and those gadgets of Section 5 that are small enough for the SAT-solver to handle

(that they implement the claimed specs and have the correct dimensions and wire offsets). The script allows the user to choose between three different SAT encodings of the local rule of Game of Life (a divide-and-conquer encoding with no auxiliary variables, an encoding based on sorting networks, and an encoding based on a merge operation) to reduce the likelihood of errors in the encoding resulting in false positives.

References

- [1] Gilles Audemard and Laurent Simon. Glucose 4.1, 2016.
- [2] Robert Berger. The undecidability of the domino problem. *Mem. Amer. Math. Soc. No.*, 66, 1966. 72 pages.
- [3] Bruno Durand, Leonid A. Levin, and Alexander Shen. Complex tilings. *Journal of Symbolic Logic*, 73:593–613, 6 2008.
- [4] Bruno Durand and Zsuzsanna Róka. The game of life: universality revisited. In *Cellular automata (Saissac, 1996)*, volume 460 of *Math. Appl.*, pages 51–74. Kluwer Acad. Publ., Dordrecht, 1999.
- [5] Noam D. Elkies. The still-life density problem and its generalizations. *arXiv preprint math/9905194*, 1999.
- [6] Martin Gardner. Mathematical Games: The Fantastic Combinations of John Conway’s New Solitaire Game “Life”. *Scientific American*, 223(4):120–123, 1970.
- [7] hengxin (https://cstheory.stackexchange.com/users/12739/hengxin). Curious about computer-assisted NP-completeness proofs. Theoretical Computer Science Stack Exchange. URL:https://cstheory.stackexchange.com/q/29340 (version: 2015-02-02).
- [8] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.
- [9] Emmanuel Jeandel. The periodic domino problem revisited. *Theoretical computer science*, 411(44-46):4010–4016, 2010.
- [10] Emmanuel Jeandel and Michaël Rao. An aperiodic set of 11 Wang tiles. *Adv. Comb.*, pages Paper No. 1, 37, 2021.
- [11] N. Johnston and D. Greene. *Conway’s Game of Life: Mathematics and Construction*. Lulu.com, 2022.
- [12] Lifewiki. https://conwaylife.com/wiki/Main_Page. (version: 2022-11-15).
- [13] Dale Myers. Nonrecursive tilings of the plane. II. *J. Symbolic Logic*, 39:286–294, 1974.

- [14] Nicolas Ollinger. Intrinsically universal cellular automata. *EPTCS*, 1:199–204, 2009.
- [15] Lifewiki: Omniprimitive. <https://conwaylife.com/wiki/Omniperiodic>. (version: 2023-08-09).
- [16] Paul Rendell. Turing universality of the game of life. In *Collision-based computing*, pages 513–539. Springer, 2002.
- [17] Tom Rokicki and Andrew Trevorrow. Golly 4.2, 2022.
- [18] Ville Salo and Ilkka Törmä. Gardens of Eden in the Game of Life. In *Automata and complexity—essays presented to Eric Goles on the occasion of his 70th birthday*, volume 42 of *Emerg. Complex. Comput.*, pages 399–415. Springer, Cham, 2022.
- [19] Ville Salo and Ilkka Törmä. gol-backward-comp. <https://github.com/ilkka-torma/gol-backward-comp>, 2022. GitHub repository.
- [20] Ville Salo and Ilkka Törmä. gol-preim. <https://github.com/ilkka-torma/gol-preim>, 2022. GitHub repository.

A The composite gadgets corresponding to gate tiles

In this section we list the parts of all the 450×450 gadgets of Section 5 that simulate the tiles of T . Each gadget consists of one 90×90 gadget surrounded by at most four 180×90 gadgets (appropriately rotated) around it that correct the phases of the wires. To get a 90×90 middle block, we connect a basic gadget of Section 4 to the sides of the block with some inverters attached, making sure the inverters charge “toward” the gadget. We use outgoing arrows to mark the middle positions of where wires go out (if present), i.e. the 29th and 30th row and column. Gate tile \square is clearly implemented by the all-zero pattern, so we omit it. The other gadgets are shown in figures 11 through 21. Figures 22 through 30 show the 180×90 gadgets that charge wires and allow us to choose their phase at the boundary. One can also insert the charged wire gadgets directly between two 90×90 tile gadgets, as we did in Section 6.

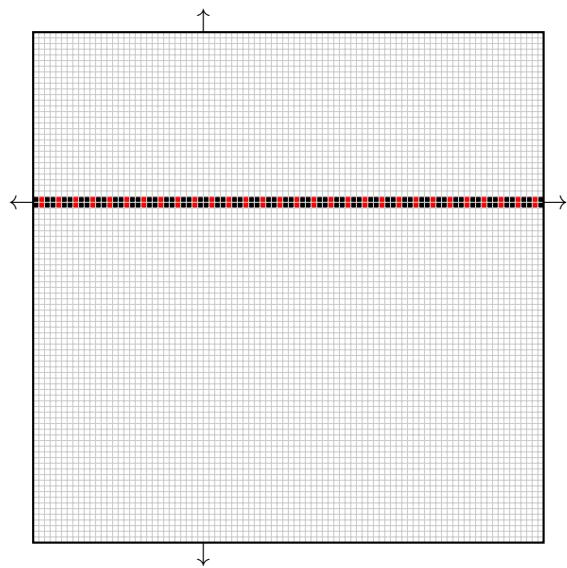


Figure 11: The gadget for \square .

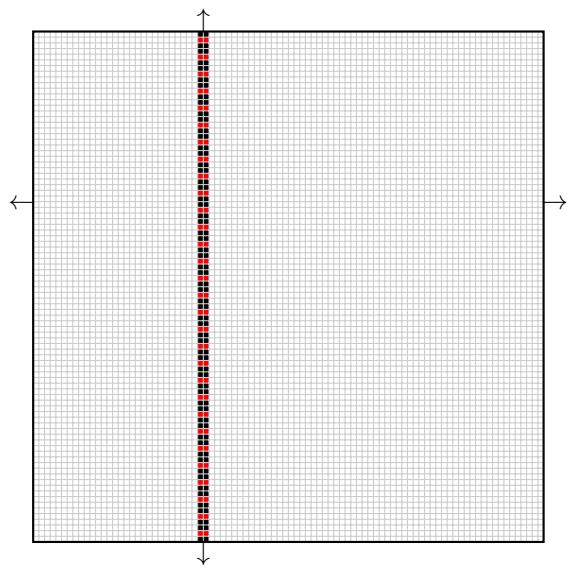


Figure 12: The gadget for $\square\Box$.

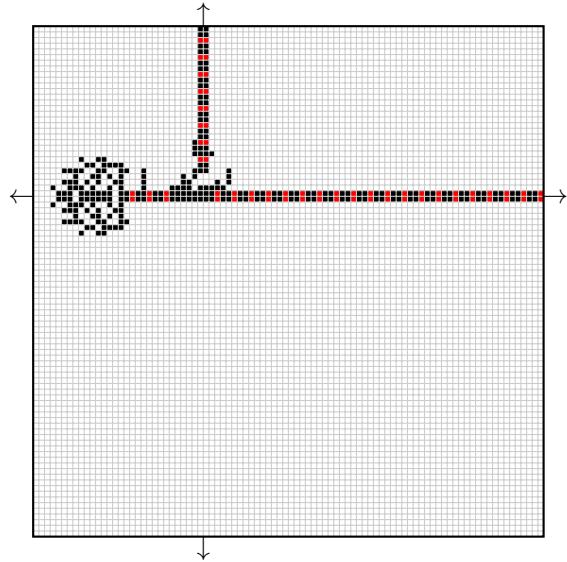


Figure 13: The gadget for \Box .

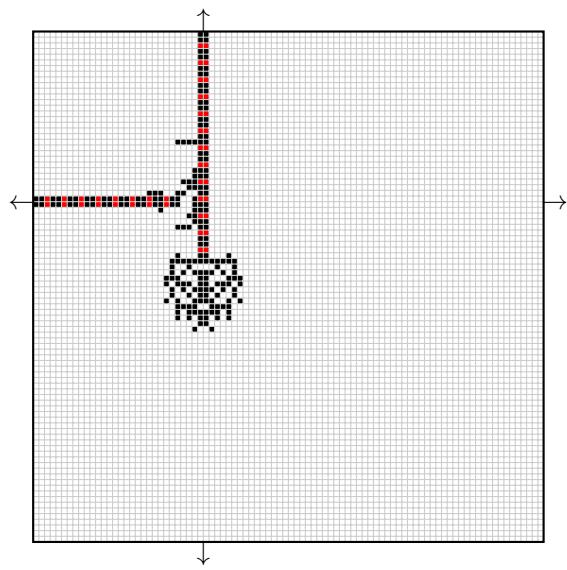


Figure 14: The gadget for \Box .

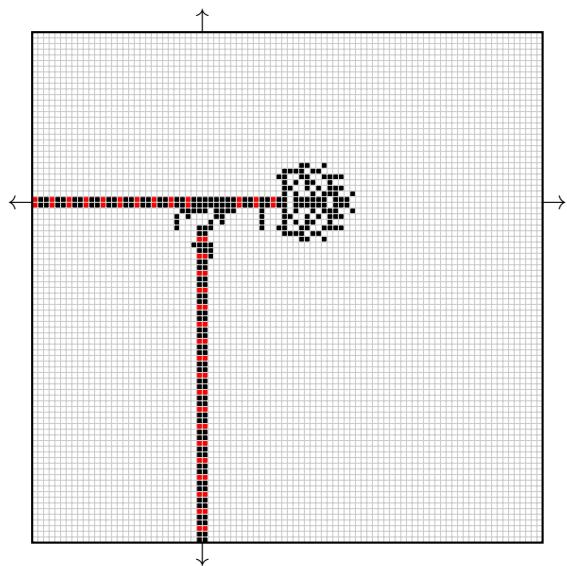


Figure 15: The gadget for \square .

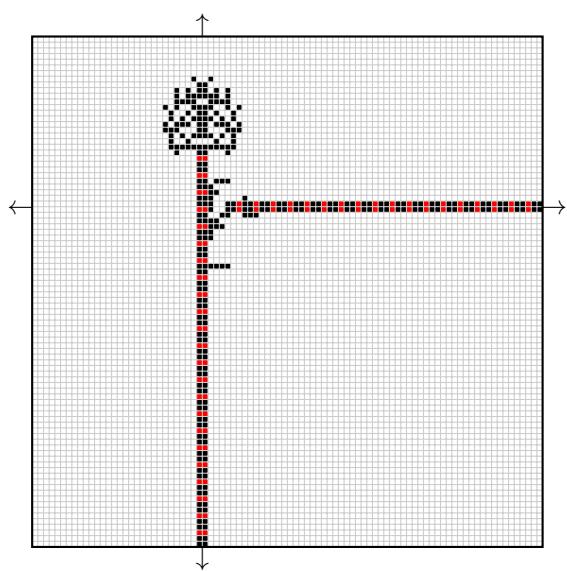


Figure 16: The gadget for \square .

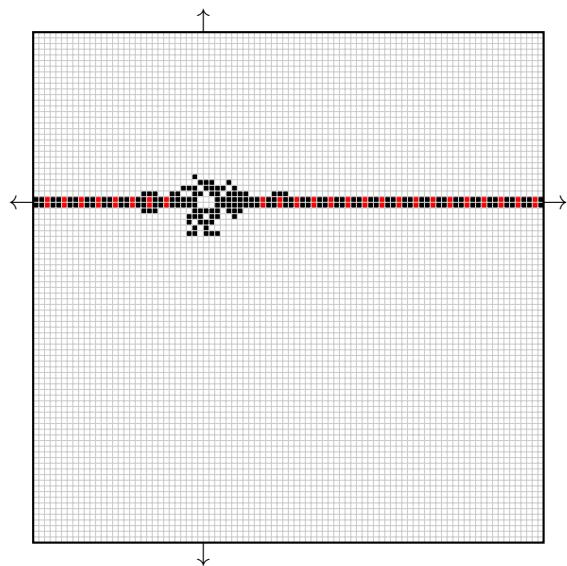


Figure 17: The gadget for \boxplus .

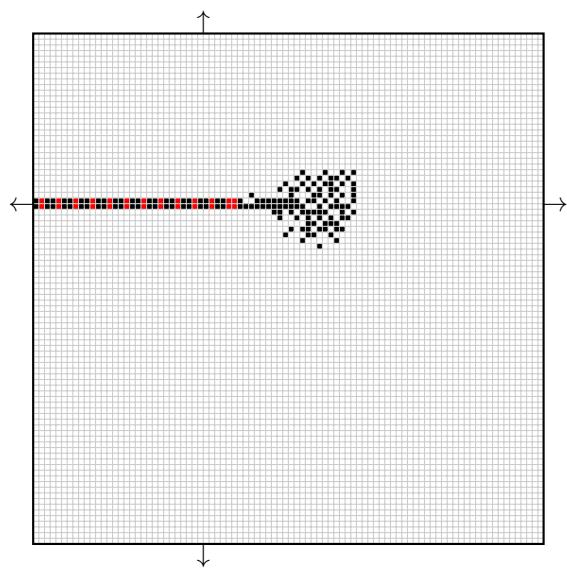


Figure 18: The gadget for \boxtimes .

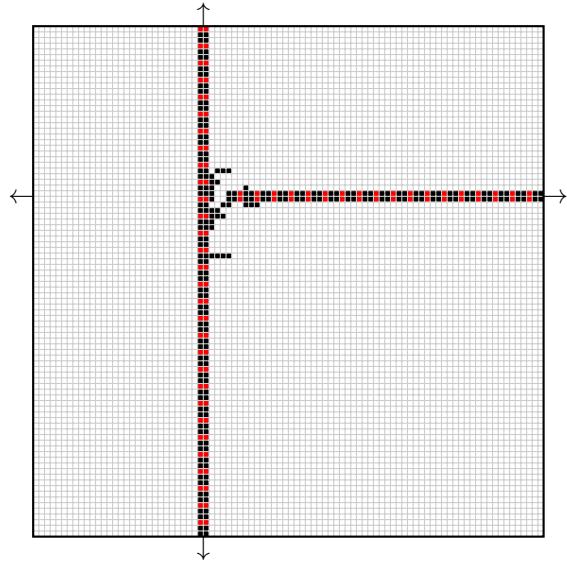


Figure 19: The gadget for \Box .

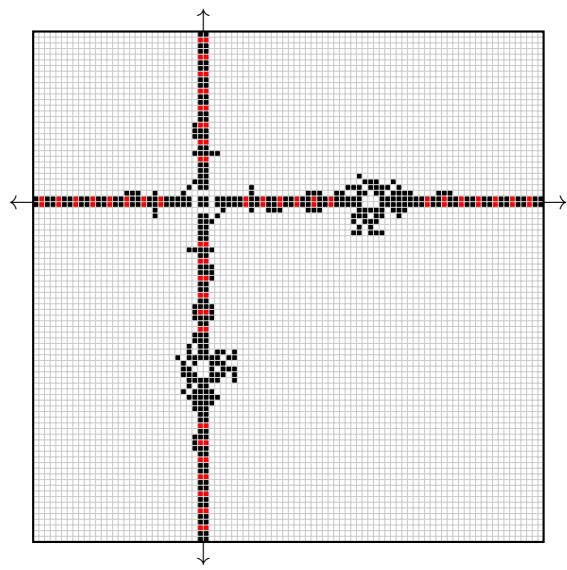


Figure 20: The gadget for \boxplus .

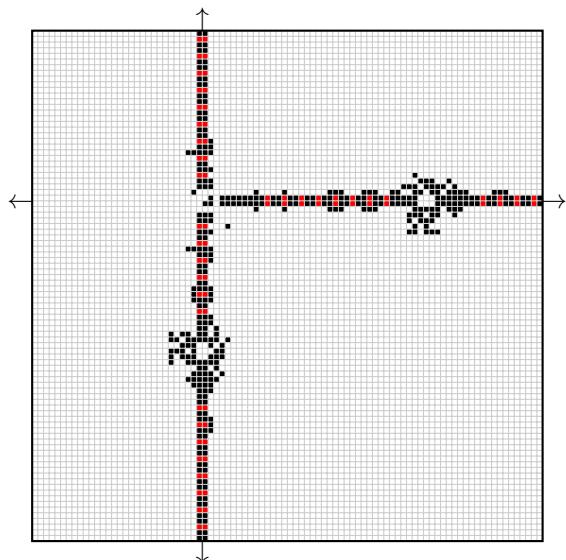


Figure 21: The gadget for \mathbb{D} .

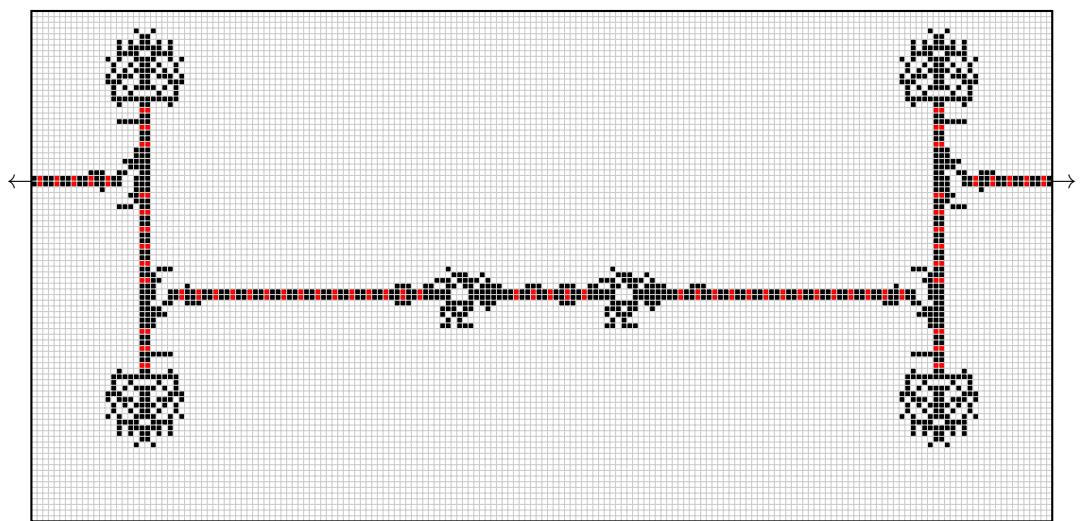


Figure 22: Horizontal charged wire gadget with neutral phases.

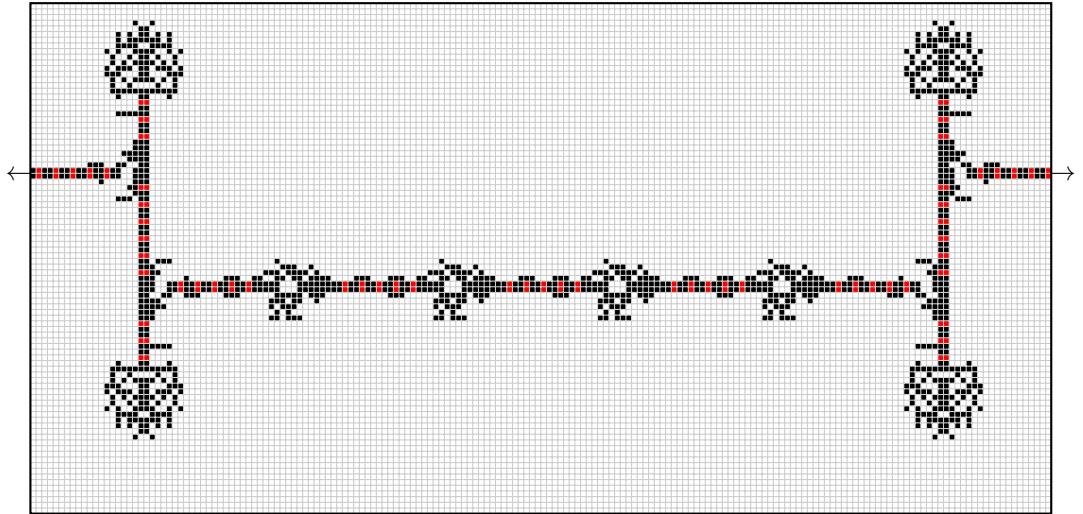


Figure 23: Horizontal charged wire gadget with neutral and east-shifted phases.

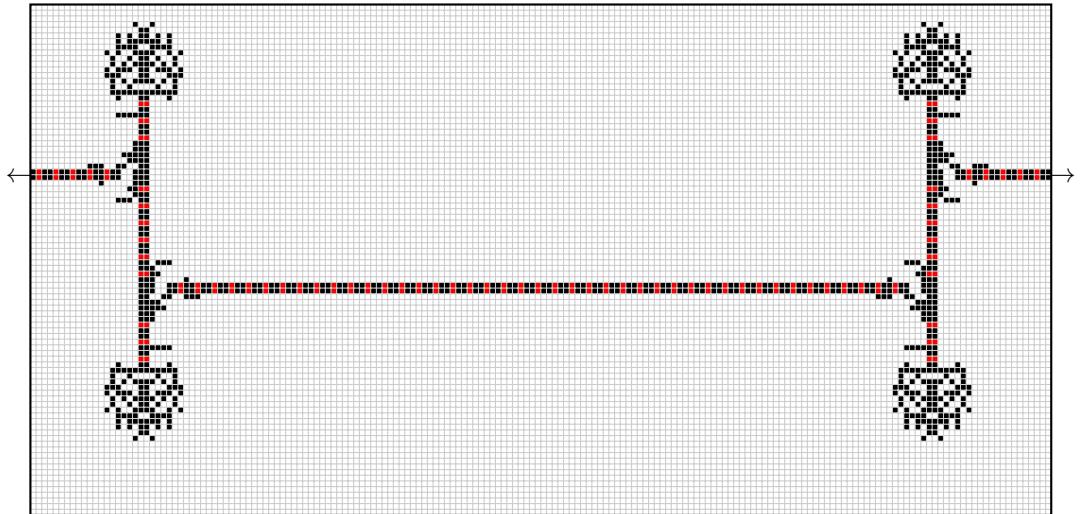


Figure 24: Horizontal charged wire gadget with neutral and west-shifted phases.

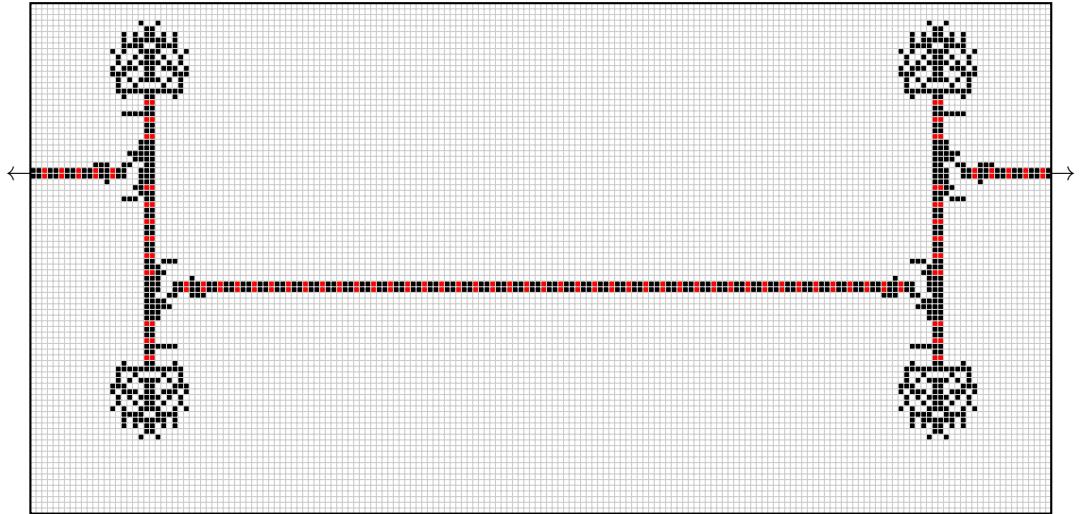


Figure 25: Horizontal charged wire gadget with east-shifted and neutral phases.

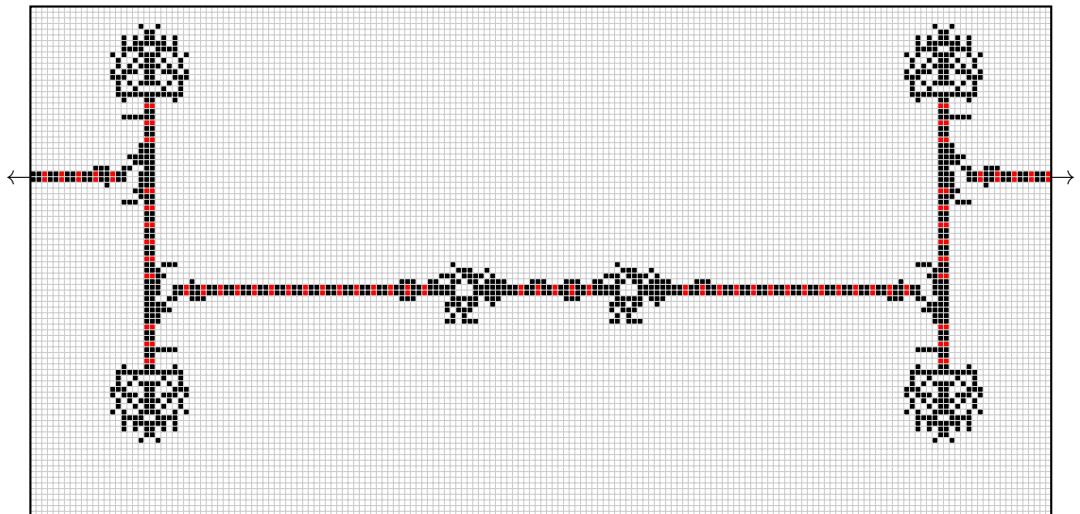


Figure 26: Horizontal charged wire gadget with east-shifted phases.

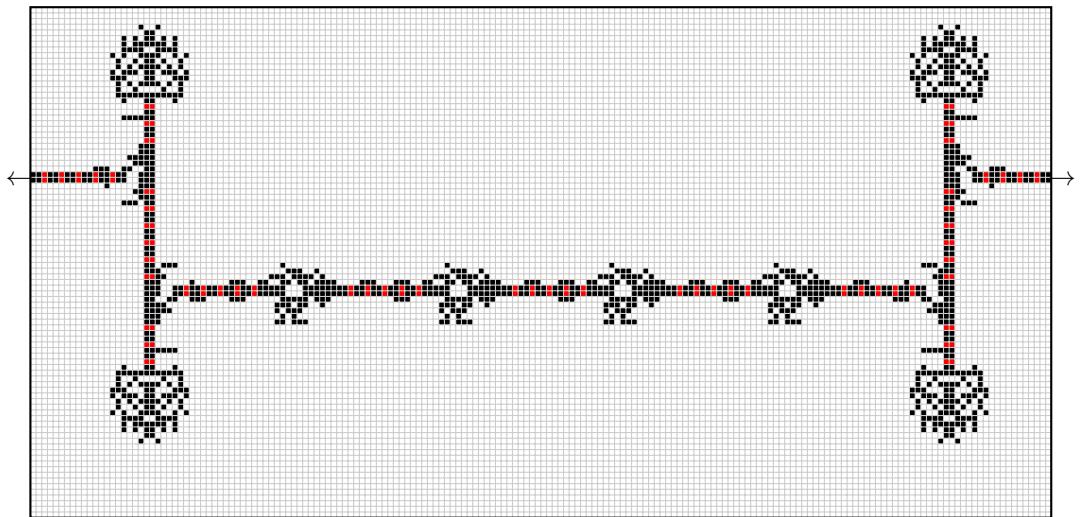


Figure 27: Horizontal charged wire gadget with east-shifted and west-shifted phases.

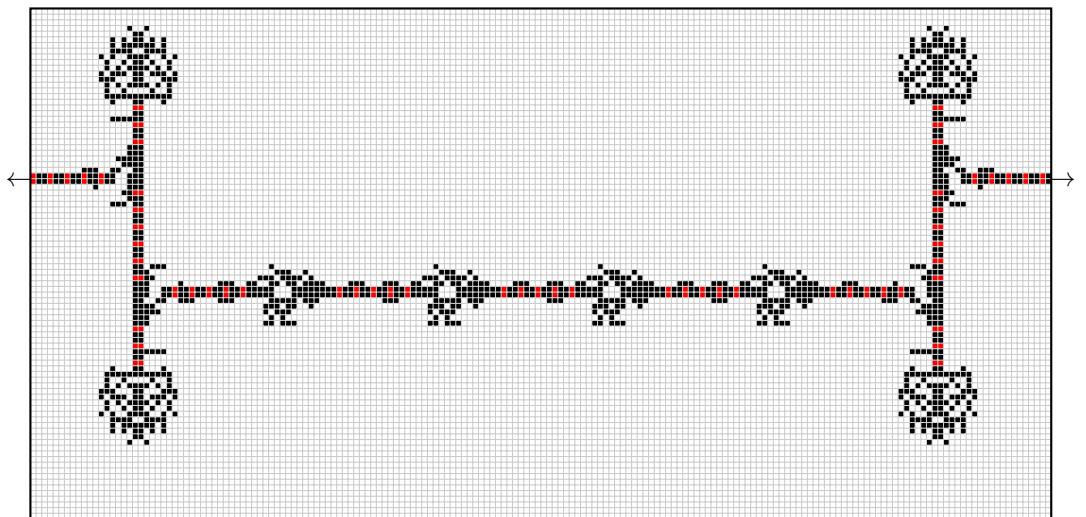


Figure 28: Horizontal charged wire gadget with west-shifted and neutral phases.

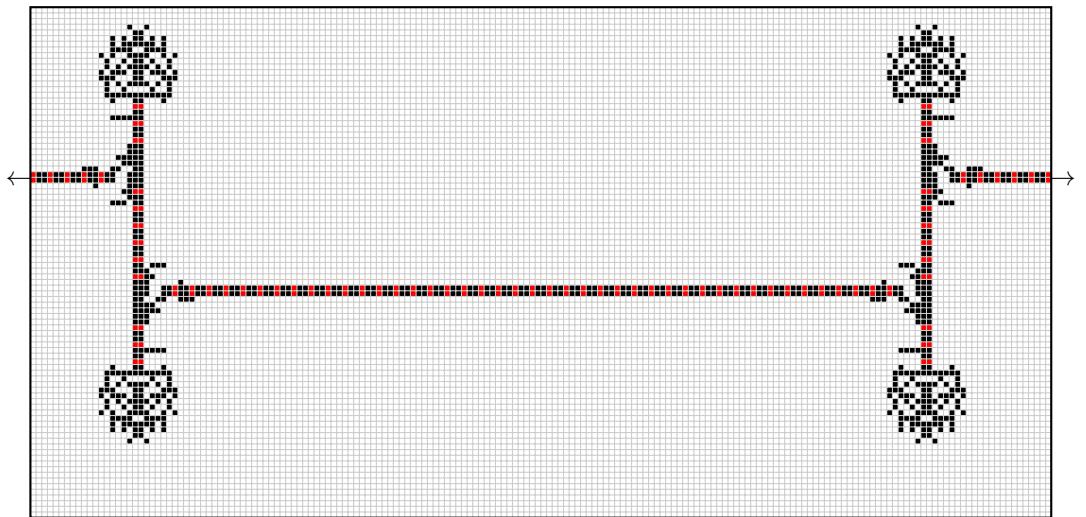


Figure 29: Horizontal charged wire gadget with west-shifted and east-shifted phases.

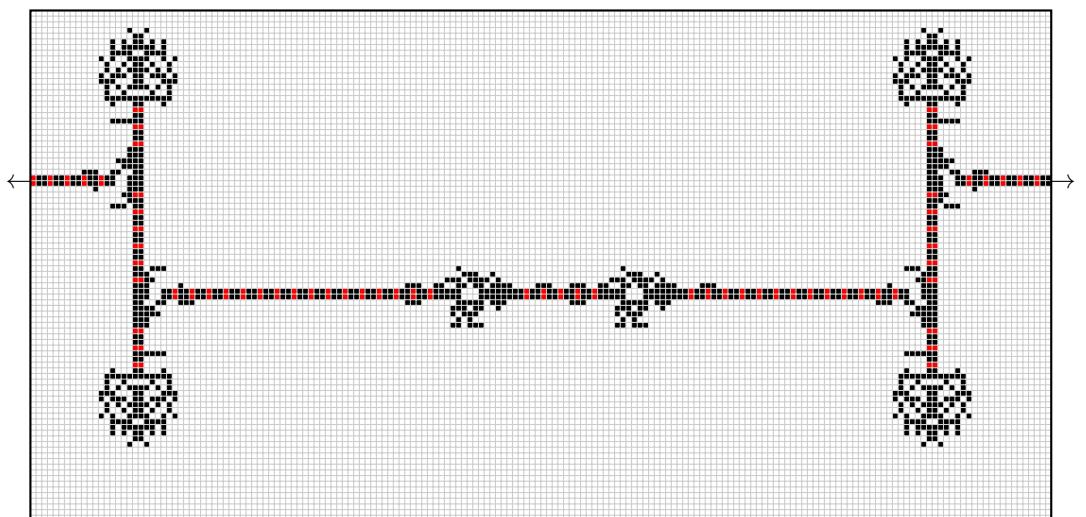


Figure 30: Horizontal charged wire gadget with west-shifted phases.