Reversible Cellular Automata

Jarkko Kari

Department of Mathematics and Statistics

University of Turku, Finland

Topics

- Introduction, definition and examples
- Hedlund's theorem
- Balance, orphans and Garden-Of-Eden

- Universality in reversible CA
- Local reversibility vs. global reversibility
- CA on periodic configurations
- Decidability questions

Cellular Automata (CA): Introduction

Cellular automata are an old model of computation. They are investigated

- **in physics** as discrete models of physical systems,
- **in computer science** as models of massively parallel computation under the realistic constraints of locality and uniformity,
- **in mathematics** as endomorphisms of the full shift in the context of symbolic dynamics.

Cellular automata possess several fundamental properties of the physical world: they are

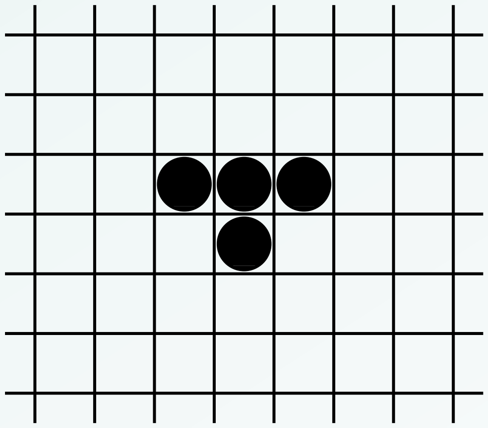
- **massively parallel**,
- **homogeneous** in time and space,
- all interactions are **local**,
- **time reversibility** and **conservation laws** can be obtained by choosing the local update rule properly.

Example: the **Game-of-Life** by John Conway.

- Infinite checker-board whose squares (=cells) are colored black (=**alive**) or white (=**dead**).
- At each discrete time step each cell counts the number of living cells surrounding it, and based on this number determines its new state.
- All cells change their state simultaneously.

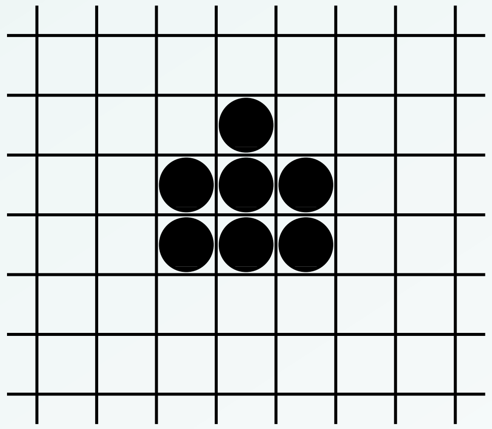
The local update rule asks each cell to check the present states of the eight surrounding cells.

- If the cell is **alive** then it stays alive (survives) iff it has two or three live neighbors. Otherwise it dies of loneliness or overcrowding.
- If the cell is **dead** then it becomes alive iff it has exactly three living neighbors.



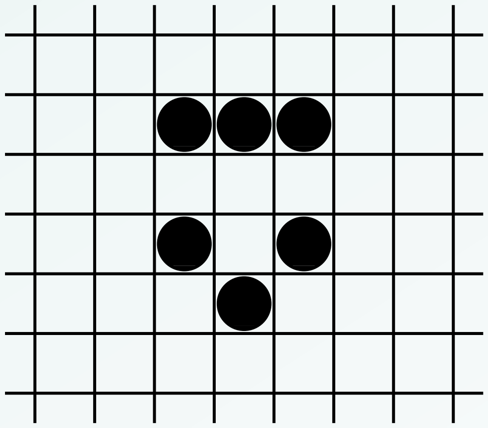
The local update rule asks each cell to check the present states of the eight surrounding cells.

- If the cell is **alive** then it stays alive (survives) iff it has two or three live neighbors. Otherwise it dies of loneliness or overcrowding.
- If the cell is **dead** then it becomes alive iff it has exactly three living neighbors.



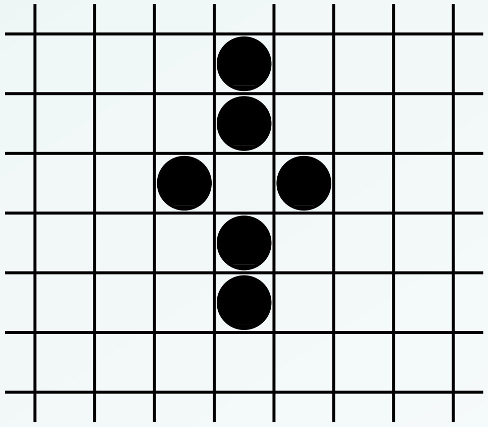
The local update rule asks each cell to check the present states of the eight surrounding cells.

- If the cell is **alive** then it stays alive (survives) iff it has two or three live neighbors. Otherwise it dies of loneliness or overcrowding.
- If the cell is **dead** then it becomes alive iff it has exactly three living neighbors.

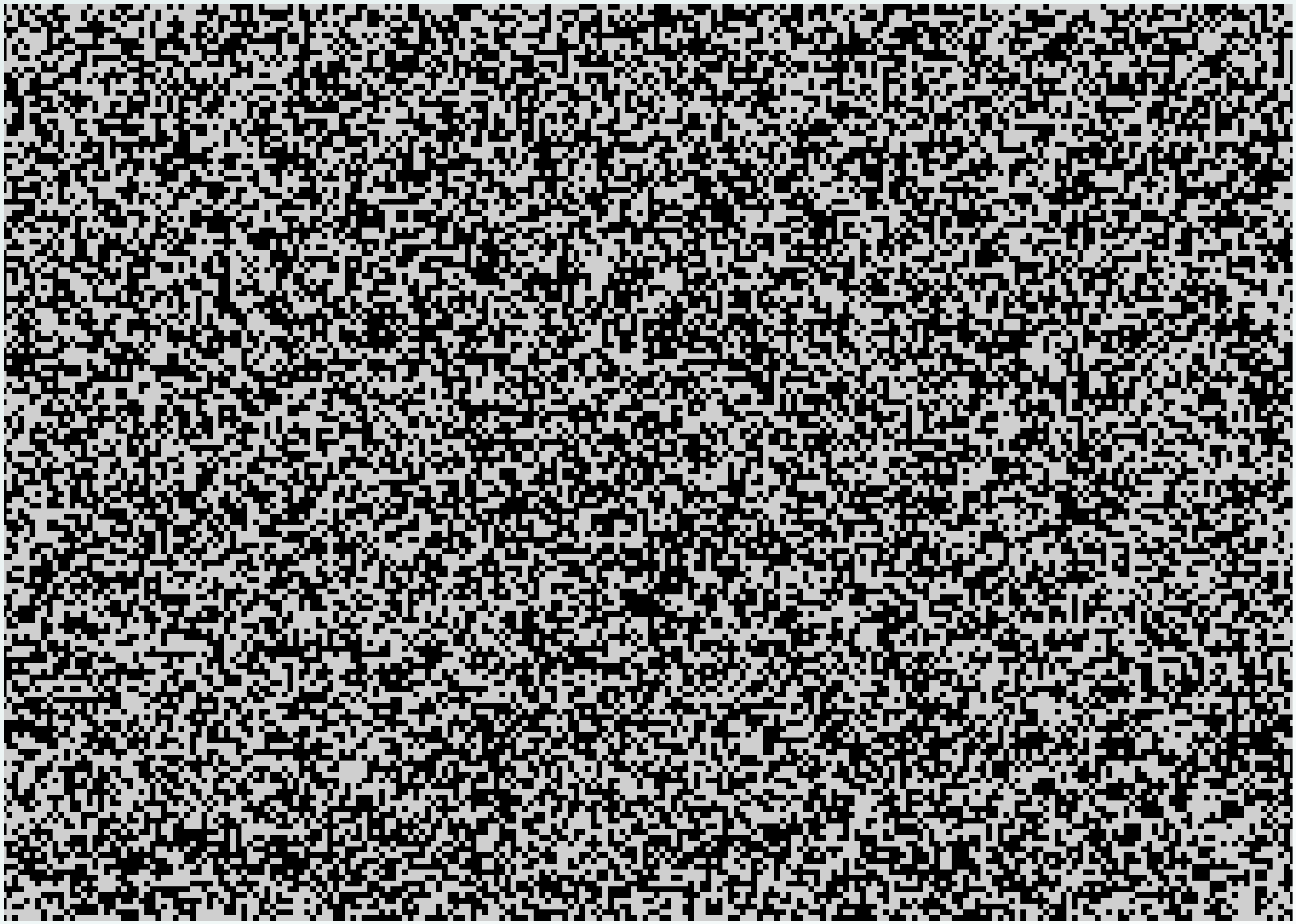


The local update rule asks each cell to check the present states of the eight surrounding cells.

- If the cell is **alive** then it stays alive (survives) iff it has two or three live neighbors. Otherwise it dies of loneliness or overcrowding.
- If the cell is **dead** then it becomes alive iff it has exactly three living neighbors.

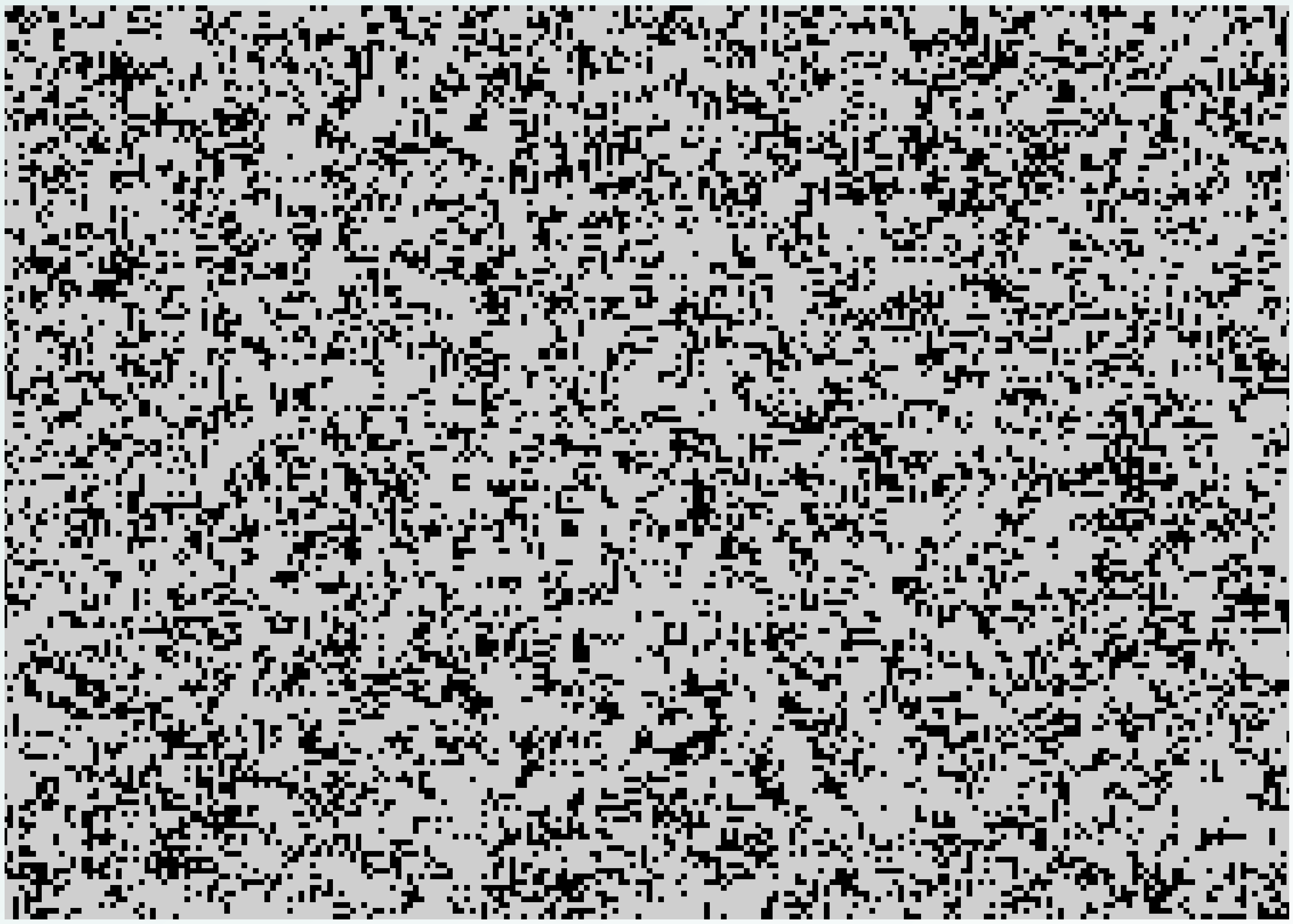


A typical snapshot of a time evolution in Game-of-Life:



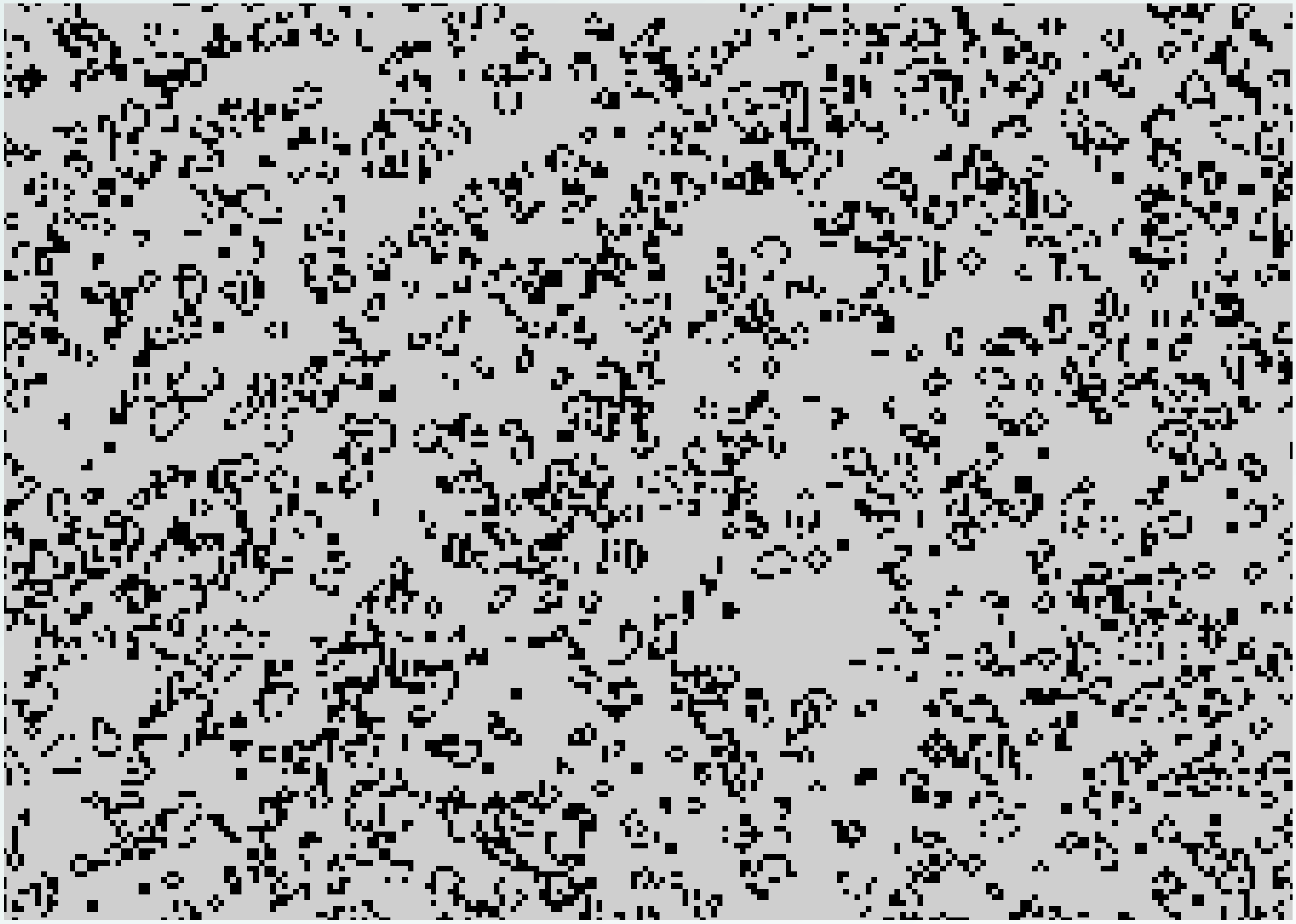
Initial uniformly random configuration.

A typical snapshot of a time evolution in Game-of-Life:



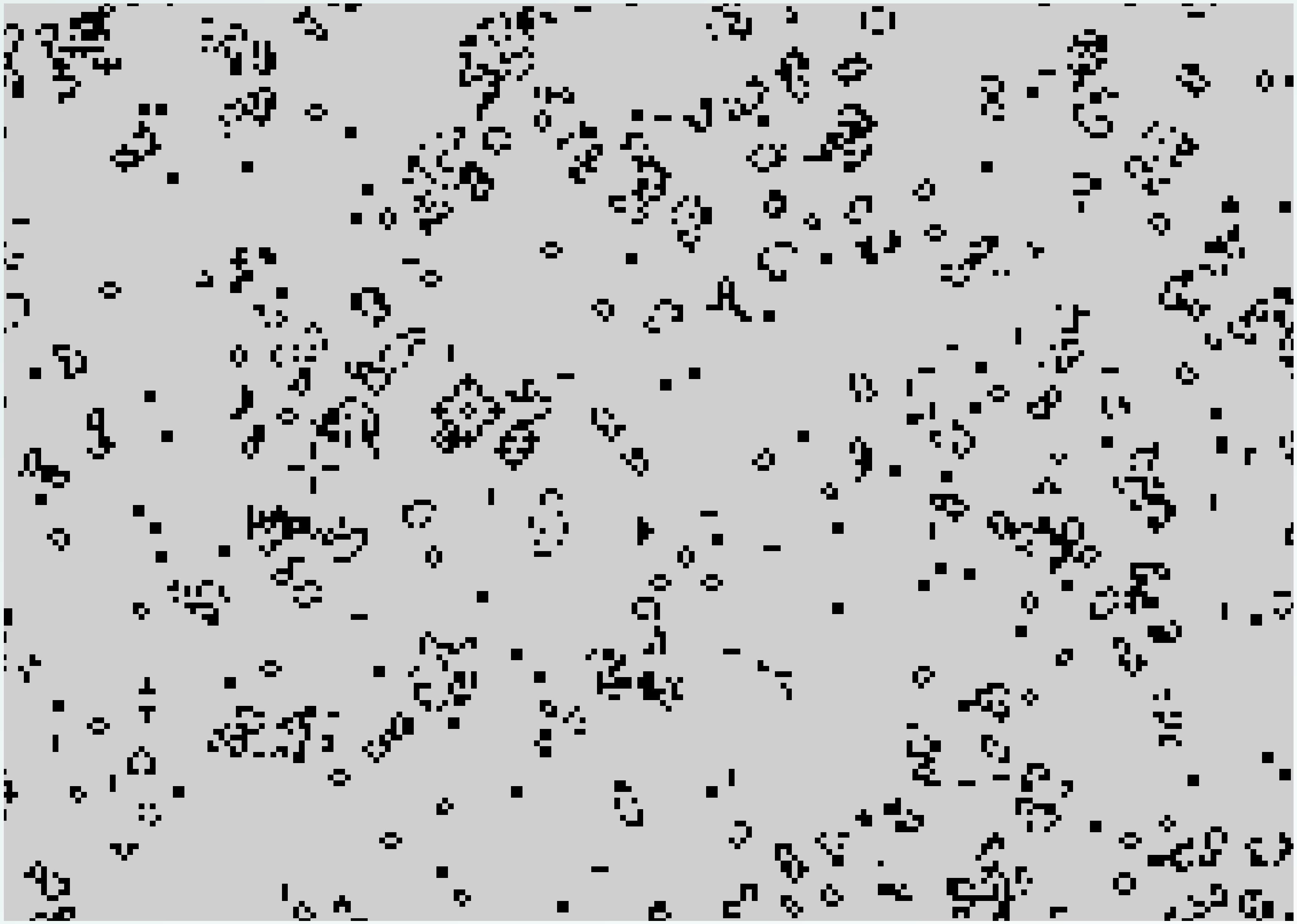
The next generation after all cells applied the update rule.

A typical snapshot of a time evolution in Game-of-Life:



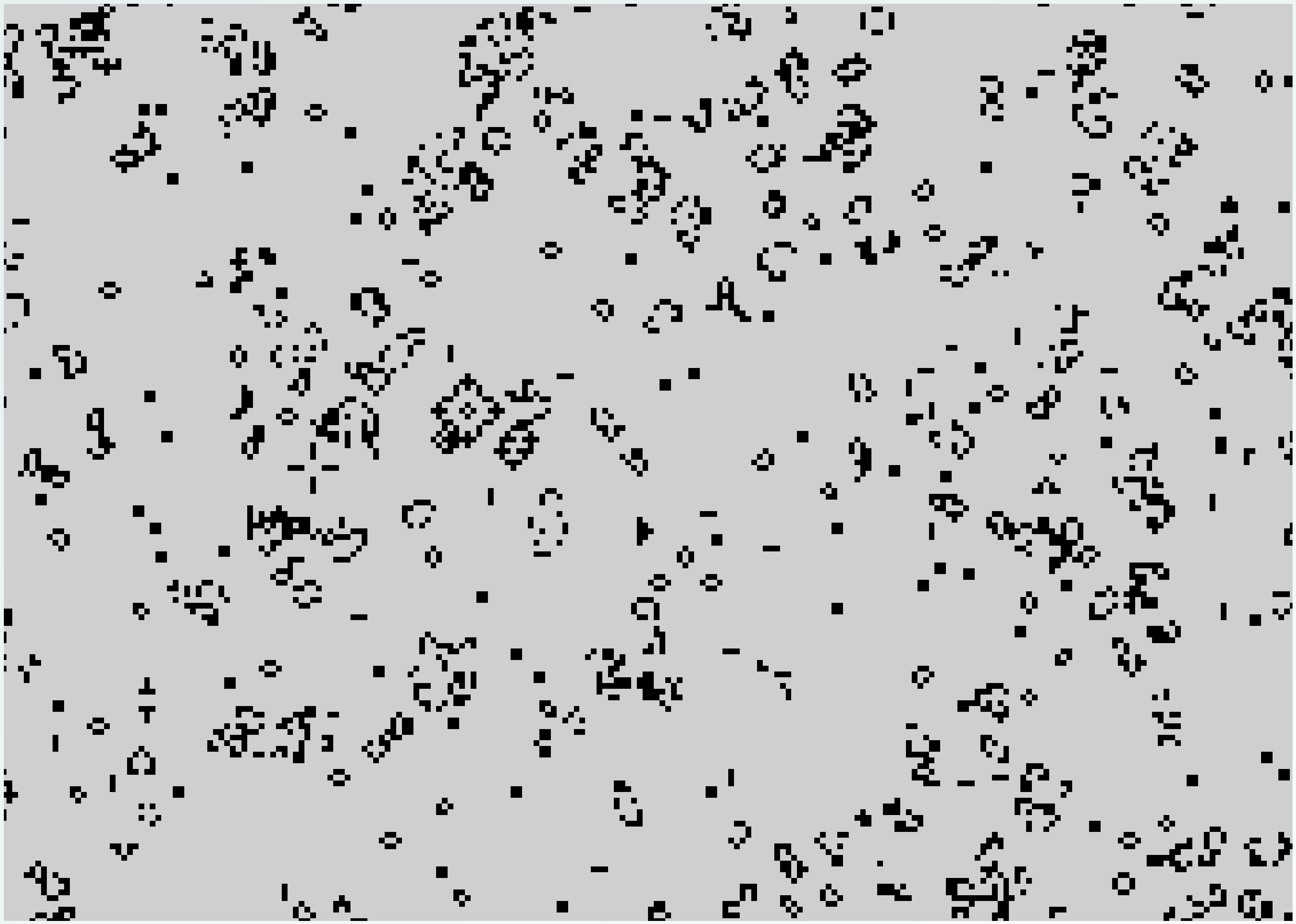
Generation 10

A typical snapshot of a time evolution in Game-of-Life:



Generation 100

A typical snapshot of a time evolution in Game-of-Life:



GOL is a computationally universal two-dimensional CA.

Another famous universal CA: **rule 110** by S.Wolfram.

A one-dimensional CA with binary state set $\{0, 1\}$, i.e. a two-way infinite sequence of 0's and 1's.

Each cell is updated based on its old state and the states of its left and right neighbors as follows:

$$\begin{array}{ccc}
111 & \longrightarrow & 0 \\
110 & \longrightarrow & 1 \\
101 & \longrightarrow & 1 \\
100 & \longrightarrow & 0 \\
011 & \longrightarrow & 1 \\
010 & \longrightarrow & 1 \\
001 & \longrightarrow & 1 \\
000 & \longrightarrow & 0
\end{array}$$

Another famous universal CA: **rule 110** by S.Wolfram.

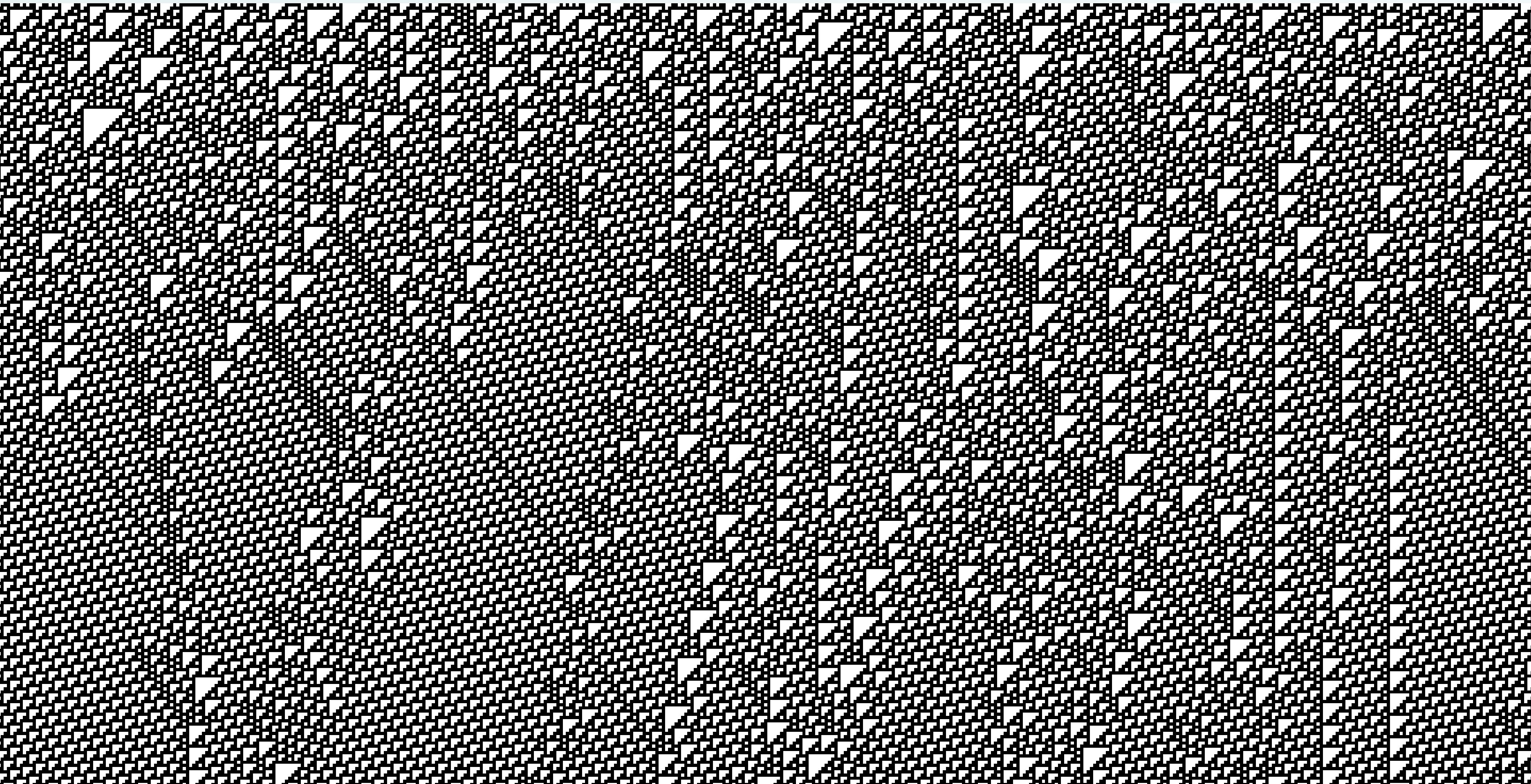
A one-dimensional CA with binary state set $\{0, 1\}$, i.e. a two-way infinite sequence of 0's and 1's.

Each cell is updated based on its old state and the states of its left and right neighbors as follows:

$$\begin{array}{ccc}
111 & \longrightarrow & 0 \\
110 & \longrightarrow & 1 \\
101 & \longrightarrow & 1 \\
100 & \longrightarrow & 0 \\
011 & \longrightarrow & 1 \\
010 & \longrightarrow & 1 \\
001 & \longrightarrow & 1 \\
000 & \longrightarrow & 0
\end{array}$$

110 is the **Wolfram number** of this CA rule.

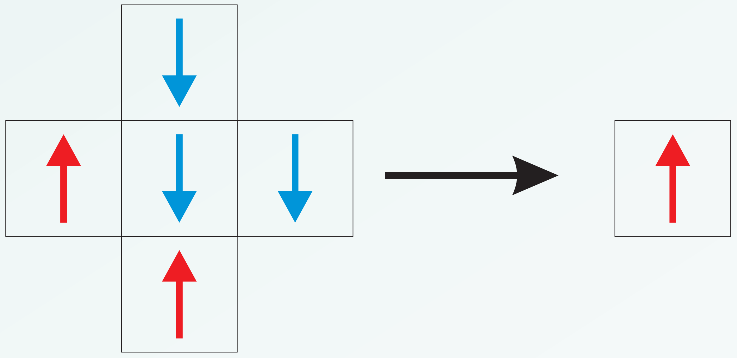
Space-time diagram is a pictorial representation of a time evolution in one-dimensional CA, where space and time are represented by the horizontal and vertical direction:



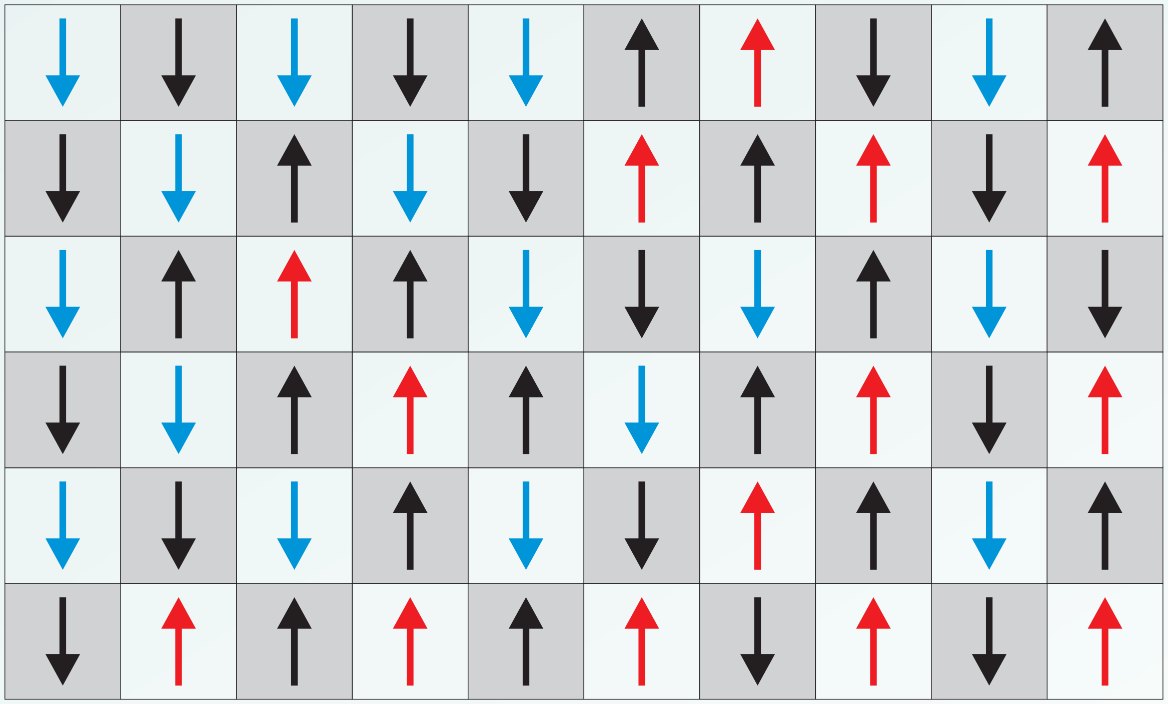
Game-of-Life and Rule 110 are **irreversible**: Configurations may have several pre-images.

Two-dimensional **Q2R** Ising model by G.Vichniac (1984) is an example of a time-reversible cellular automaton.

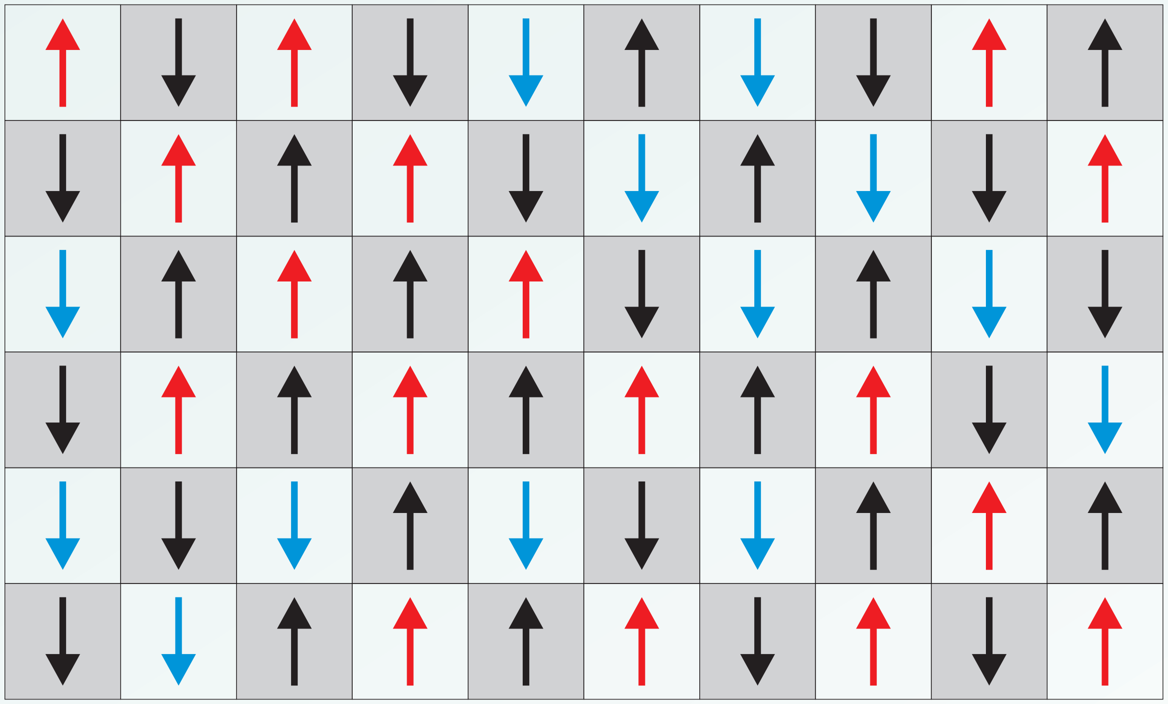
Each cell has a spin that is directed either up or down. The direction of a spin is swapped if and only if among the four immediate neighbors there are exactly two cells with spin up and two cells with spin down:



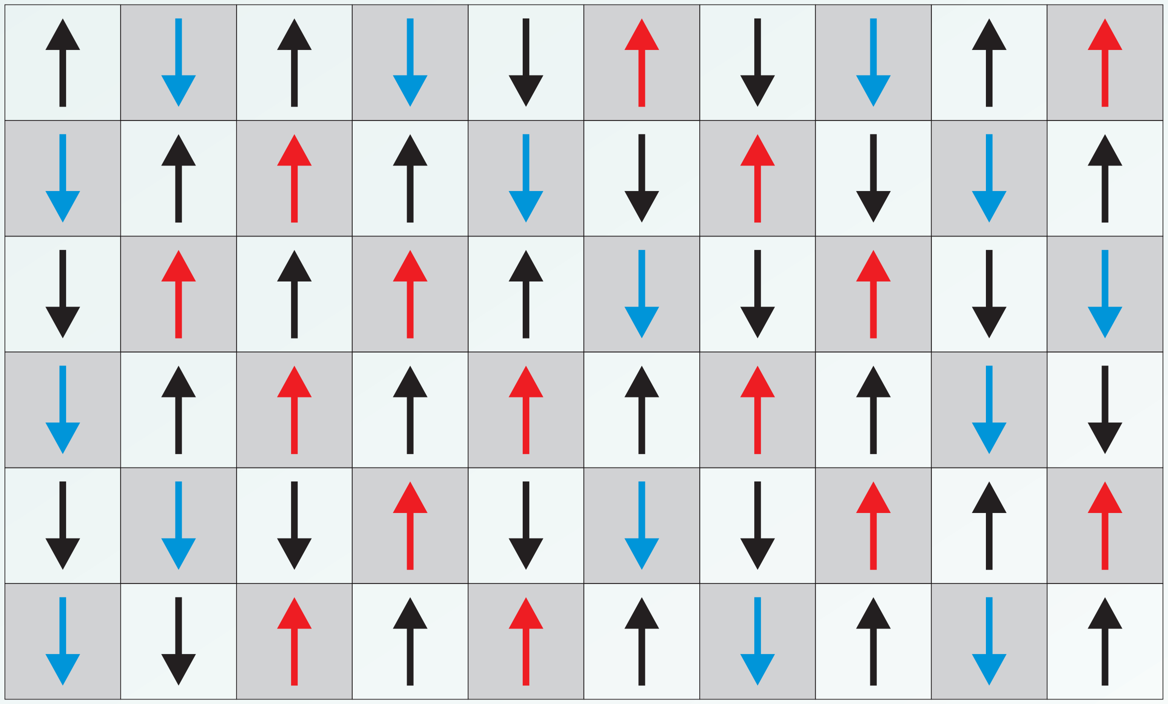
The twist that makes the Q2R rule reversible: Color the space as a checker-board. On even time steps only update the spins of the white cells and on odd time steps update the spins of the black cells.



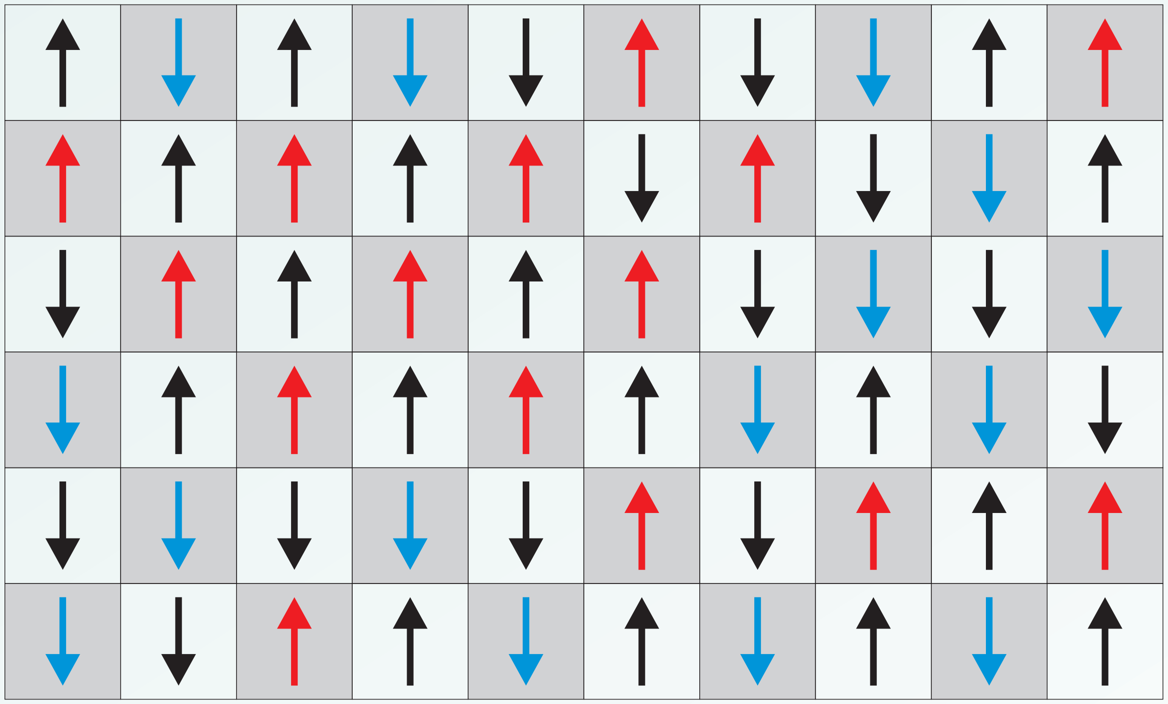
The twist that makes the Q2R rule reversible: Color the space as a checker-board. On even time steps only update the spins of the white cells and on odd time steps update the spins of the black cells.



The twist that makes the Q2R rule reversible: Color the space as a checker-board. On even time steps only update the spins of the white cells and on odd time steps update the spins of the black cells.

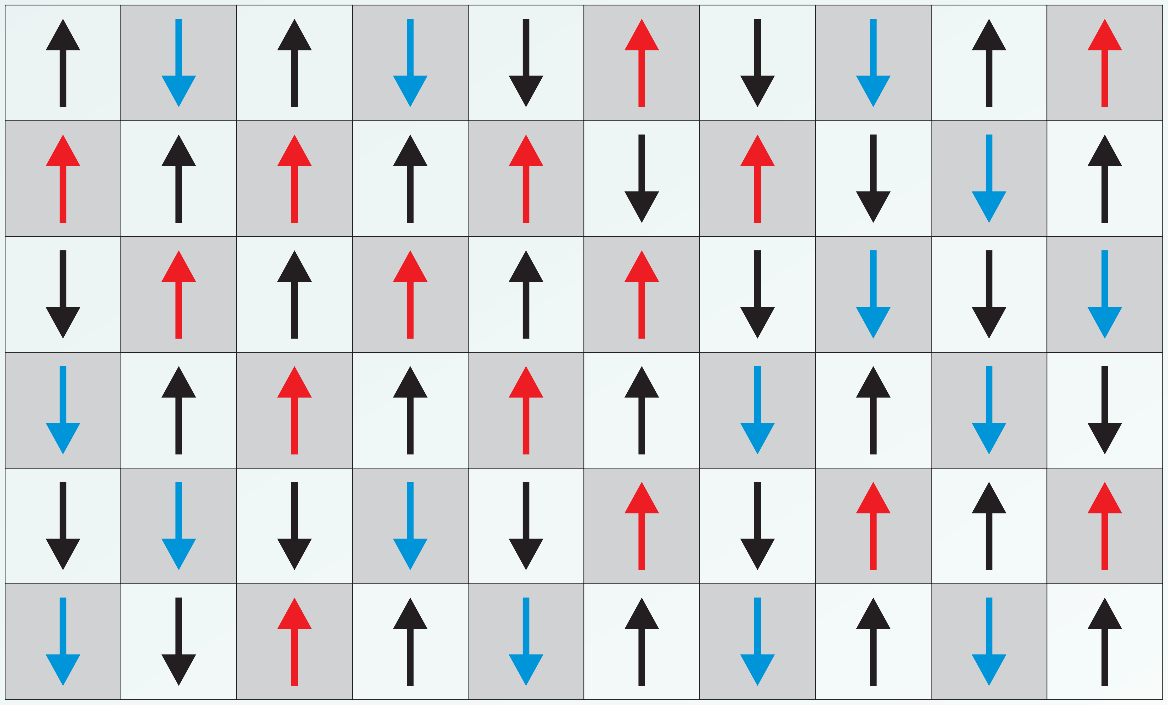


The twist that makes the Q2R rule reversible: Color the space as a checker-board. On even time steps only update the spins of the white cells and on odd time steps update the spins of the black cells.

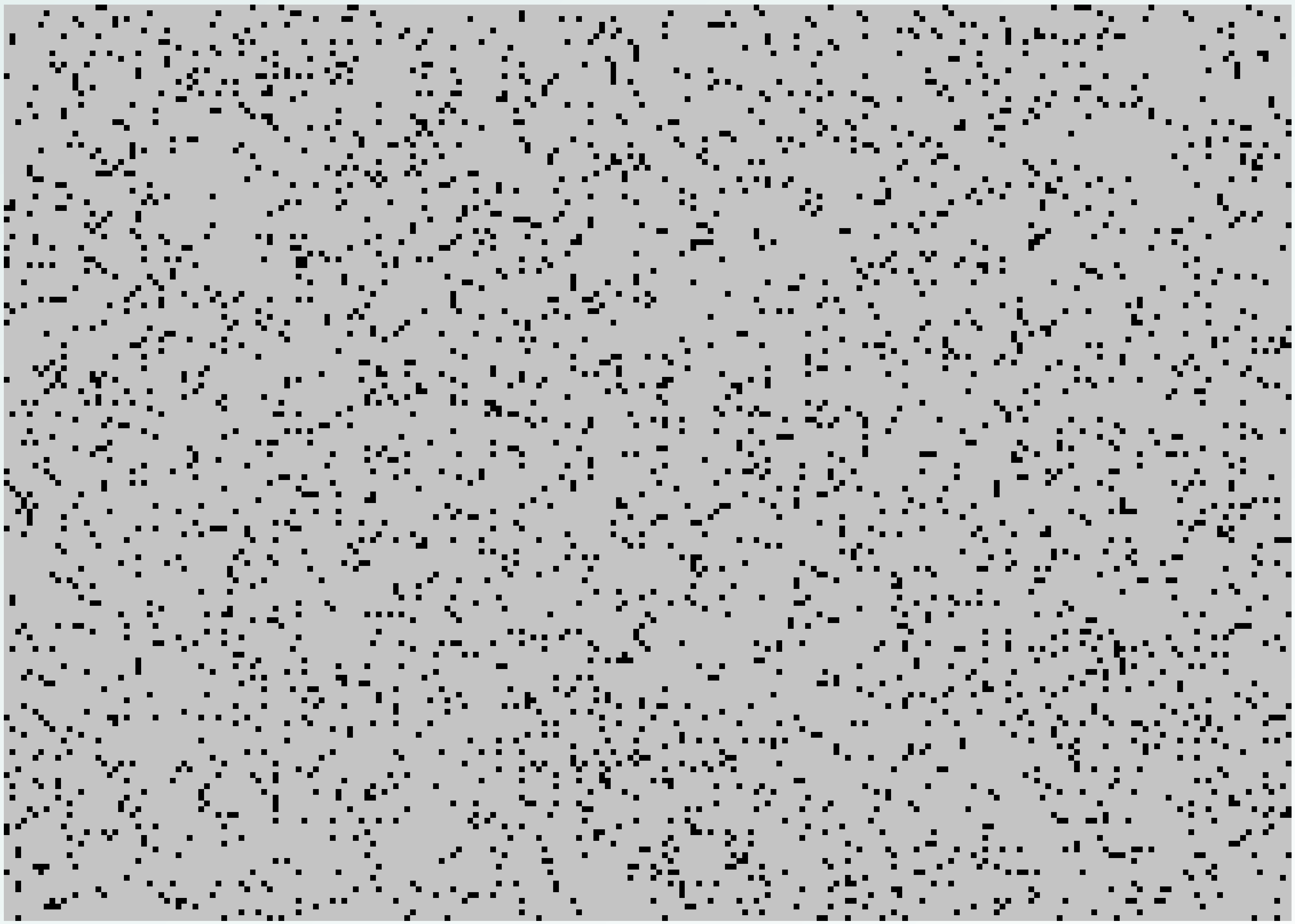


Q2R is **reversible**: The same rule (applied again on squares of the same color) reconstructs the previous generation.

Q2R rule also exhibits a local **conservation law**: The number of neighbors with opposite spins remains constant over time.

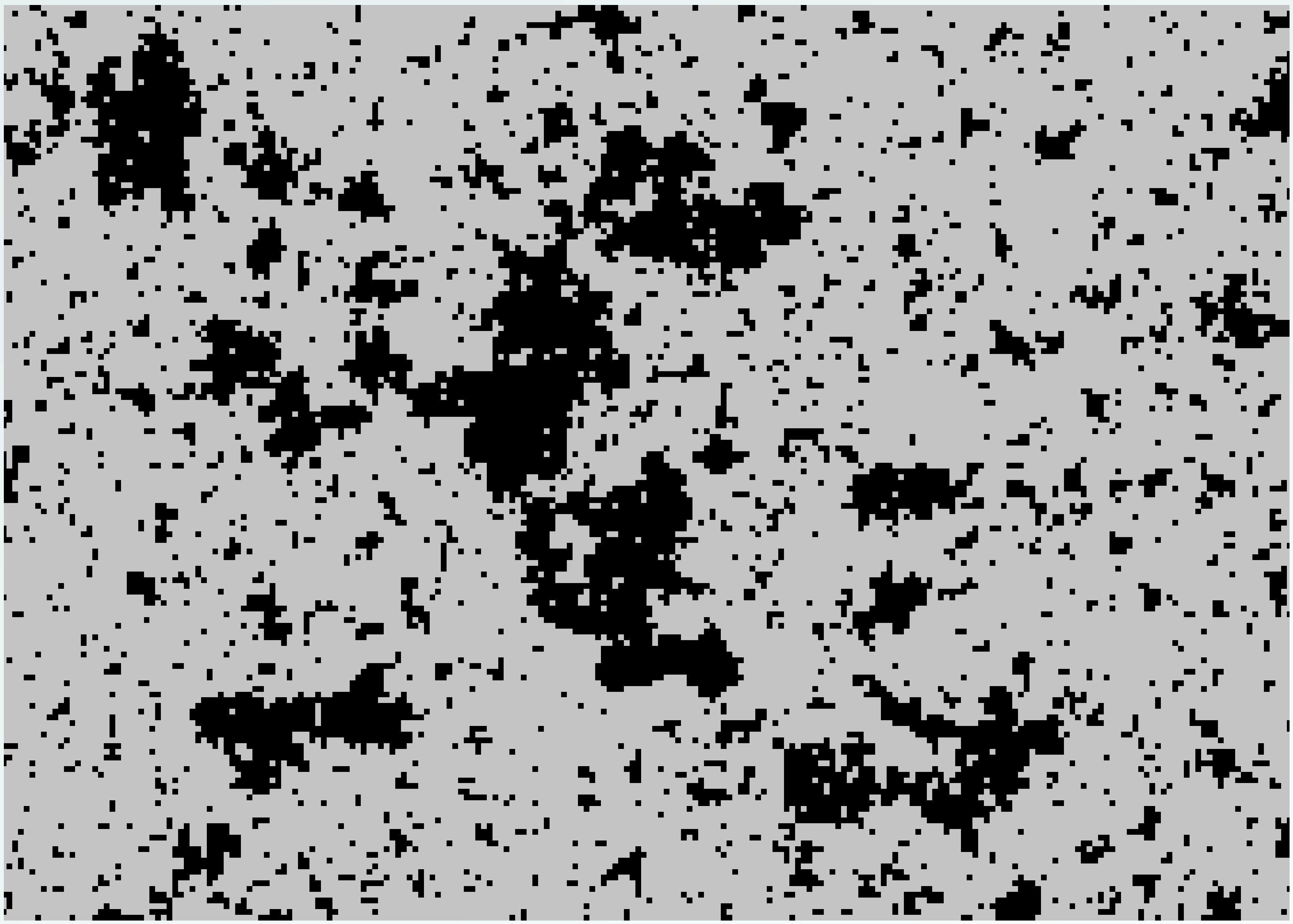


Evolution of Q2R from an uneven random distribution of spins:



Initial random configuration with 8% spins up.

Evolution of Q2R from an uneven random distribution of spins:



One million steps. The length of the B/W boundary is invariant.

General definition of d-dimensional CA

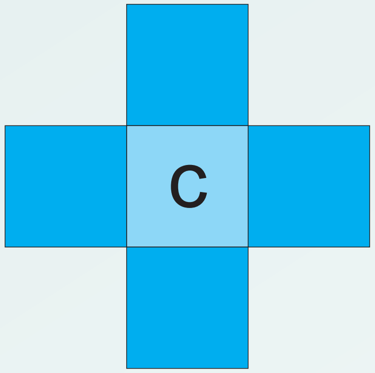
- Finite **state set** S.
- **Configurations** are elements of $S^{\mathbb{Z}^d}$, i.e., functions $\mathbb{Z}^d \longrightarrow S$ assigning states to cells,
- A **neighborhood vector**

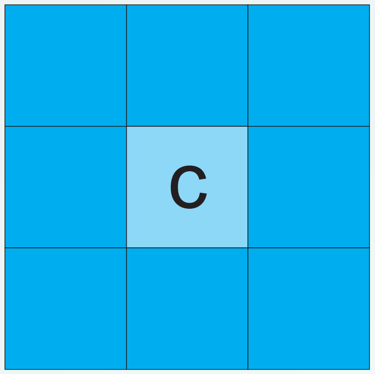
  $$N = (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n)$$

  is a vector of n distinct elements of $\mathbb{Z}^d$ that provide the relative offsets to neighbors.
- The **neighbors** of a cell at location $\vec{x} \in \mathbb{Z}^d$ are the n cells at locations

  $$\vec{x} + \vec{x}_i, \text{ for } i = 1, 2, \ldots, n.$$

Typical two-dimensional neighborhoods:





Von Neumann

neighborhood

$\{(0,0),(\pm 1,0),(0,\pm 1)\}$

Moore

neighborhood

$\{-1,0,1\}\times\{-1,0,1\}$

The **local rule** is a function

$$f : S^n \longrightarrow S$$

where n is the size of the neighborhood.

State $f(a_1, a_2, \ldots, a_n)$ is the new state of a cell whose n neighbors were at states $a_1, a_2, \ldots, a_n$ one time step before.

The **global dynamics** of the CA: Configuration c becomes in one time step the configuration e where, for all $\vec{x} \in \mathbb{Z}^d$,

$$e(\vec{x}) = f(c(\vec{x} + \vec{x}_1), c(\vec{x} + \vec{x}_2), \ldots, c(\vec{x} + \vec{x}_n)).$$

The transformation

$$G : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$$

that maps $c \mapsto e$ is the **CA function**.

A CA is

- **injective** if G is one-to-one,
- **surjective** if G is onto,
- **bijective** if G is both one-to-one and onto.

A CA is

- **injective** if G is one-to-one,
- **surjective** if G is onto,
- **bijective** if G is both one-to-one and onto.

A CA G is a **reversible** (RCA) if there is another CA function F that is its inverse, i.e.

$$G \circ F = F \circ G = \text{identity function.}$$

RCA G and F are called the **inverse automata** of each other.

Curtis-Hedlund-Lyndon -theorem

It is convenient to endow $S^{\mathbb{Z}^d}$ with a **metric** to measures distances of configurations: For all $c \neq e$,

$$d(c, e) = 2^{-n}$$

where

$$n = \min\{||\vec{x}|| \mid c(\vec{x}) \neq e(\vec{x})\}$$

is the distance from the origin to the closest cell where c and e differ.

Two configurations are close to each other if one needs to look far to see a difference in them.

In the **usual metric** on $\mathbb{R}^d$ one needs to change

into

if one want to distinguish very close points.

In the **usual metric** on $\mathbb{R}^d$ one needs to change

into

if one want to distinguish very close points.

In **our metric** on the configuration space $S^{\mathbb{Z}^d}$, a better equipment sees further away:

In the **usual metric** on $\mathbb{R}^d$ one needs to change

into

if one want to distinguish very close points.

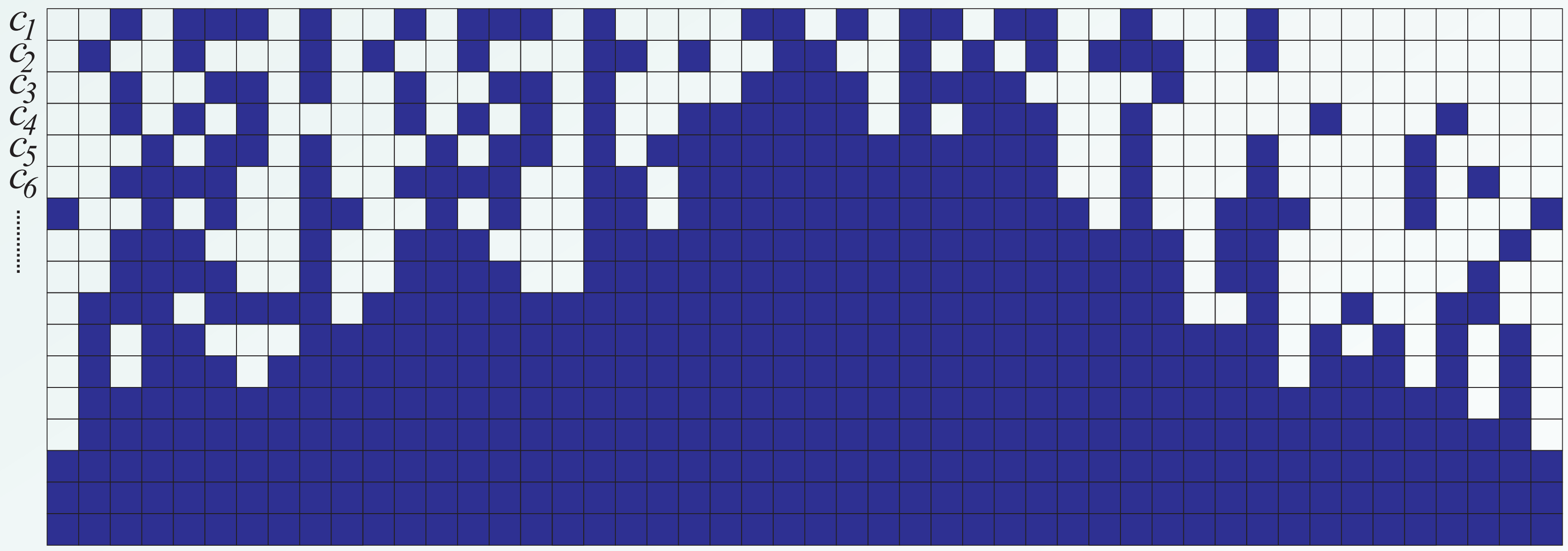
In **our metric** on the configuration space $S^{\mathbb{Z}^d}$, a better equipment sees further away:

Under this metric, space $S^{\mathbb{Z}^d}$ is **compact** : every infinite sequence $c_1, c_2, \ldots$ of configurations has a converging subsequence.

Under this metric, space $S^{\mathbb{Z}^d}$ is **compact** : every infinite sequence $c_1, c_2, \ldots$ of configurations has a converging subsequence.

A sequence $c_1, c_2, \ldots$ of configurations **converges** to $c \in S^{\mathbb{Z}^d}$ iff for all cells $\vec{x} \in \mathbb{Z}^d$ and for all sufficiently large i holds

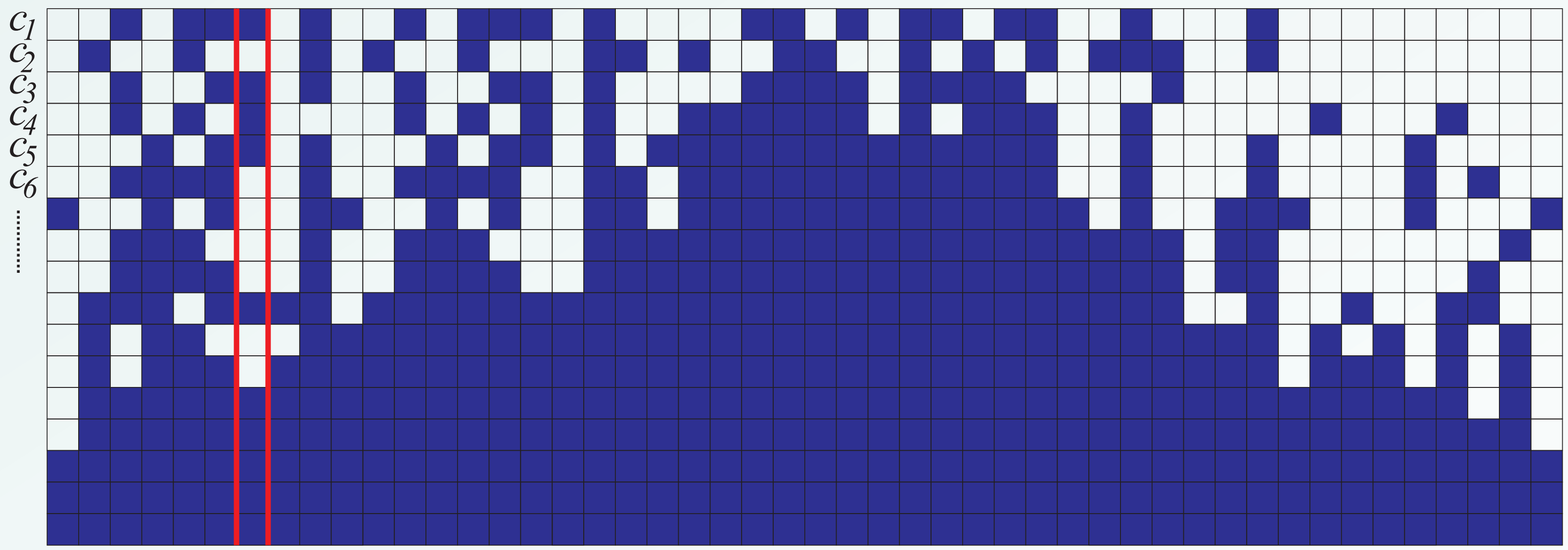
$$c_i(\vec{x}) = c(\vec{x}).$$



Under this metric, space $S^{\mathbb{Z}^d}$ is **compact** : every infinite sequence $c_1, c_2, \ldots$ of configurations has a converging subsequence.

A sequence $c_1, c_2, \ldots$ of configurations **converges** to $c \in S^{\mathbb{Z}^d}$ iff for all cells $\vec{x} \in \mathbb{Z}^d$ and for all sufficiently large i holds

$$c_i(\vec{x}) = c(\vec{x}).$$



All cellular automata are **continuous** transformations

$$S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$$

under our metric. Indeed, locality of the update rule means that if

$$c_1, c_2, \ldots$$

is a converging sequence of configurations then

$$G(c_1), G(c_2), \ldots$$

converges as well, and

$$\lim_{i\to\infty} G(c_i) = G(\lim_{i\to\infty} c_i).$$

The **translation** τ determined by vector $\vec{r} \in \mathbb{Z}^d$ is the transformation

$$S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$$

that maps $c \mapsto e$ where

$$e(\vec{x}) = c(\vec{x} - \vec{r}) \text{ for all } \vec{x} \in \mathbb{Z}^d.$$

(It is the CA whose local rule is the identity function and whose neighborhood consists of $-\vec{r}$ alone.)

Translations determined by unit coordinate vectors $(0, \ldots, 0, 1, 0 \ldots, 0)$ are called **shifts**

Since all cells of a CA use the same local rule, the CA commutes with all translations:

$$G \circ \tau = \tau \circ G.$$

We have seen that all CA are continuous, translation commuting maps $S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$.

The **Curtis-Hedlund- Lyndon theorem** from 1969 states that also the converse is true:

Theorem: A function $G : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$ is a CA function if and only if

(i) G is continuous, and

(ii) G commutes with translations.

From the Curtis-Hedlund-Lyndon -theorem we get

Corollary: A cellular automaton G is reversible if and only if it is bijective.

From the Curtis-Hedlund-Lyndon -theorem we get

Corollary: A cellular automaton G is reversible if and only if it is bijective.

Proof: If G is a reversible CA function then G is by definition bijective.

Conversely, suppose that G is a bijective CA function. Then G has an inverse function G^{-1} that clearly commutes with the shifts. The inverse function G^{-1} is also continuous because the space $S^{\mathbb{Z}^d}$ is compact. It now follows from the Curtis-Hedlund-Lyndon theorem that G^{-1} is a cellular automaton. $\square$

From the Curtis-Hedlund-Lyndon -theorem we get

Corollary: A cellular automaton G is reversible if and only if it is bijective.

The **point of the corollary** is that in bijective CA each cell can determine its previous state by looking at the current states in some bounded neighborhood around them.

Garden-Of-Eden and orphans

Configurations that do not have a pre-image are called **Garden-Of-Eden** -configurations. Only non-surjective CA have GOE configurations.

A finite pattern consists of a finite domain $D \subseteq \mathbb{Z}^d$ and an assignment

$$p : D \longrightarrow S$$

of states.

Finite pattern is called an **orphan** for CA G if every configuration containing the pattern is a GOE.

From the compactness of $S^{\mathbb{Z}^d}$ we directly get:

Proposition. Every GOE configuration contains an orphan pattern.

Non-surjectivity is hence equivalent to the existence of orphans.

Balance in surjective CA

All surjective CA have **balanced** local rules: for every $a \in S$

$$\left|f^{-1}(a)\right| = |S|^{n-1}.$$

Balance in surjective CA

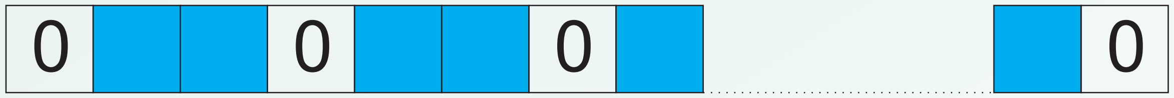
All surjective CA have **balanced** local rules: for every $a \in S$

$$\left|f^{-1}(a)\right| = |S|^{n-1}.$$

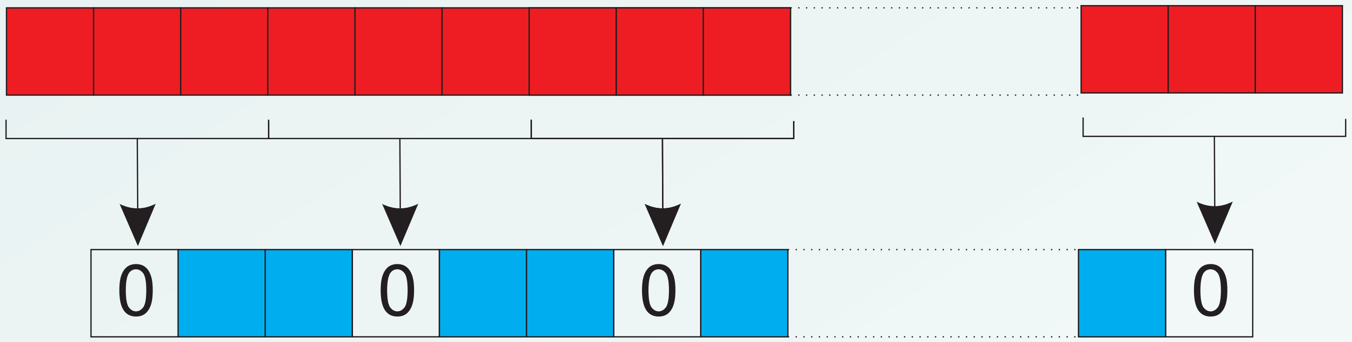
Indeed, consider a non-balanced local rule such as rule 110 where five contexts give new state 1 while only three contexts give state 0:

$$\begin{array}{ccc}
111 & \longrightarrow & 0 \\
110 & \longrightarrow & 1 \\
101 & \longrightarrow & 1 \\
100 & \longrightarrow & 0 \\
011 & \longrightarrow & 1 \\
010 & \longrightarrow & 1 \\
001 & \longrightarrow & 1 \\
000 & \longrightarrow & 0
\end{array}$$

Consider finite patterns where state 0 appears in every third position. There are $2^{2(k-1)} = 4^{k-1}$ such patterns where k is the number of 0's.



Consider finite patterns where state 0 appears in every third position. There are $2^{2(k-1)} = 4^{k-1}$ such patterns where k is the number of 0's.



A pre-image of such a pattern must consist of k segments of length three, each of which is mapped to 0 by the local rule. There are 3^k choices.

As for large values of k we have $3^k < 4^{k-1}$, there are fewer choices for the red cells than for the blue ones. Hence some pattern has no pre-image and it must be an orphan. $\square$

One can also verify directly that pattern

$$01010$$

is an orphan of rule 110. It is the shortest orphan.

Balance of the local rule is not sufficient for surjectivity. For example, the **majority** CA (Wolfram number 232) is a counter example. The local rule

$$f(a, b, c) = 1 \text{ if and only if } a + b + c \geq 2$$

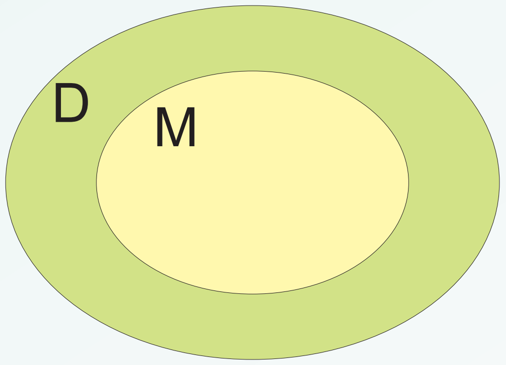
is clearly balanced, but 01001 is an orphan.

The balance property of surjective CA generalizes to finite patterns of arbitrary shape:

Theorem: Let G be surjective. Let $M, D \subseteq \mathbb{Z}^d$ be finite domains such that D contains the neighborhood of M. Then every finite pattern with domain M has the same number

$$n^{|D|-|M|}$$

of pre-images in domain D, where n is the number of states. $\square$



The balance property of surjective CA generalizes to finite patterns of arbitrary shape:

Theorem: Let G be surjective. Let $M, D \subseteq \mathbb{Z}^d$ be finite domains such that D contains the neighborhood of M. Then every finite pattern with domain M has the same number

$$n^{|D|-|M|}$$

of pre-images in domain D, where n is the number of states. $\square$

The balance property means that the uniform probability measure is **invariant** for surjective CA. (Uniform randomness is preserved by surjective CA.)

Garden-Of-Eden -theorem

Let us call configurations c_1 and c_2 **asymptotic** if the set

$$\mathit{diff}\,(c_1, c_2) = \{\vec{n} \in \mathbb{Z}^d \mid c_1(\vec{n}) \neq c_2(\vec{n})\ \}$$

of positions where c_1 and c_2 differ is finite.

A CA is called **pre-injective** if any asymptotic $c_1 \neq c_2$ satisfy $G(c_1) \neq G(c_2)$.

The **Garden-Of-Eden -theorem** by Moore (1962) and Myhill (1963) connects surjectivity with pre-injectivity.

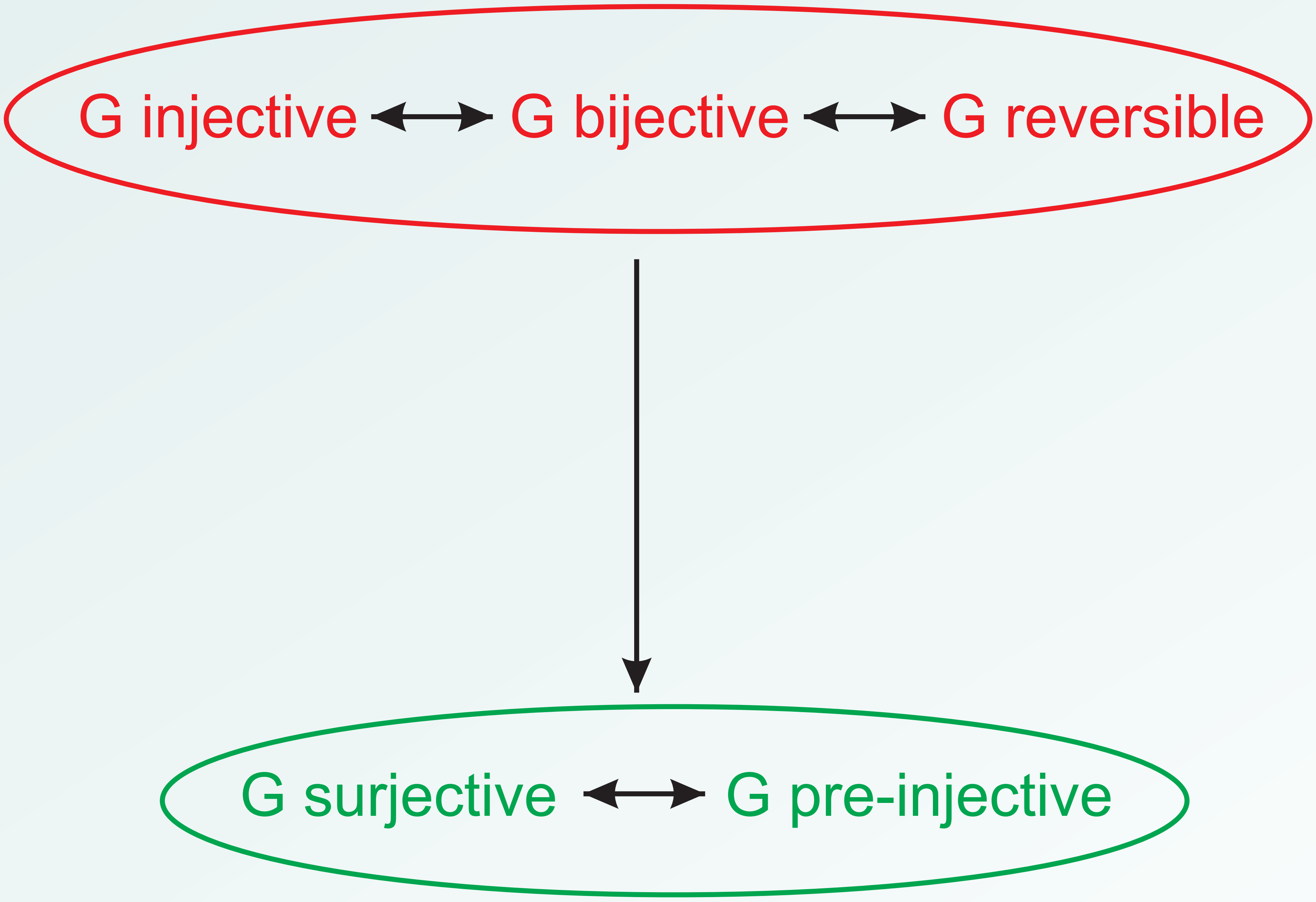
Theorem: CA G is surjective if and only if it is pre-injective.

The **Garden-Of-Eden -theorem** by Moore (1962) and Myhill (1963) connects surjectivity with pre-injectivity.

Theorem: CA G is surjective if and only if it is pre-injective.

Corollary: Every injective CA is also surjective. Injectivity, bijectivity and reversibility are equivalent concepts.

Proof: If G is injective then it is pre-injective. The claim follows from the Garden-Of-Eden -theorem. $\square$


G injective
G bijective
G reversible
G surjective
G pre-injective

Examples:

The majority rule is not surjective: asymptotic configurations

$$\ldots 0000000 \ldots \qquad \text{and} \qquad \ldots 0001000 \ldots$$

have the same image, so G is not pre-injective. Pattern

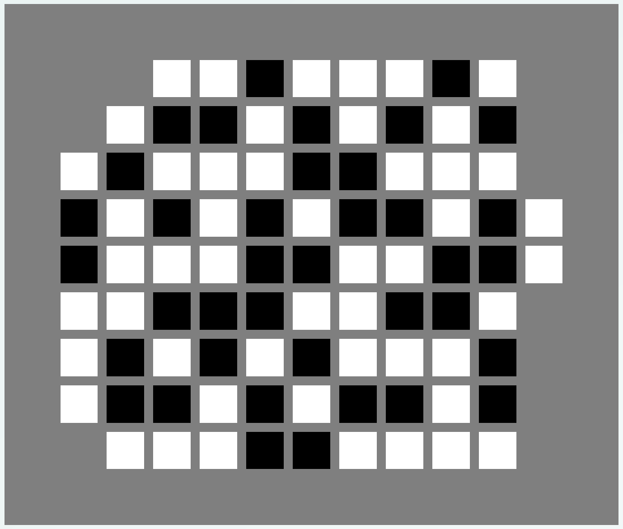
$$01001$$

is an orphan.

Examples:

In Game-Of-Life a lonely living cell dies immediately, so G is not pre-injective. GOL is hence not surjective.

Interestingly, no small orphans are known for Game-Of-Life. Currently, the smallest known orphan consists of 88 cells (50 life=white, 38 dead=black):



Steven Eker (2017)

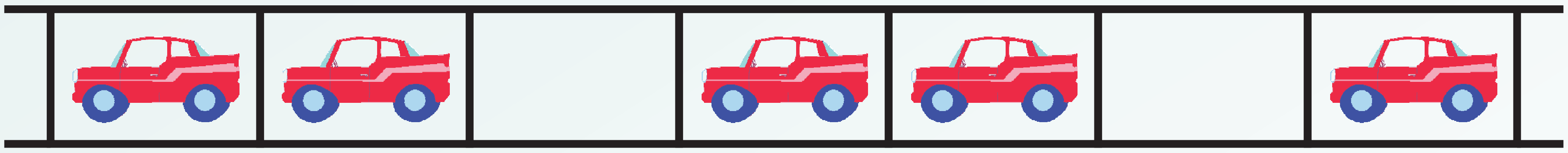
Examples:

The **Traffic CA** is the elementary CA number 226.

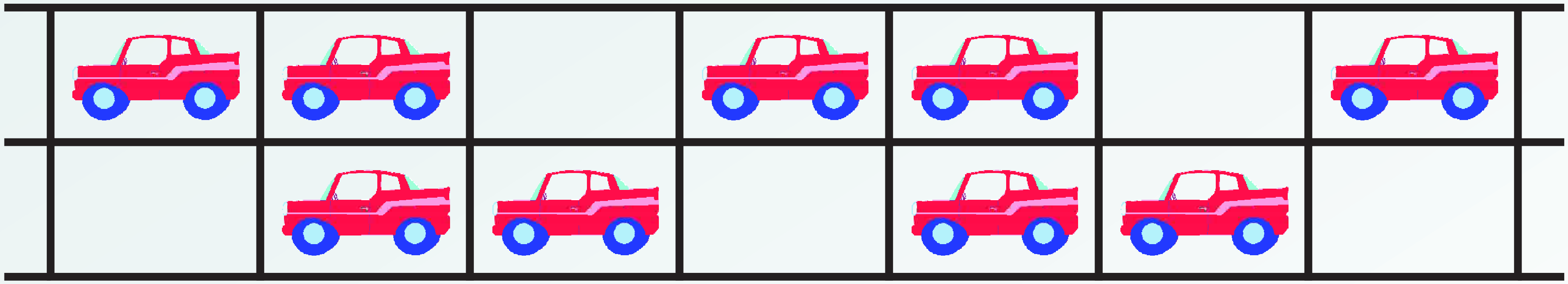
$$\begin{array}{ccc} 111 & \longrightarrow & 1 \\ 110 & \longrightarrow & 1 \\ 101 & \longrightarrow & 1 \\ 100 & \longrightarrow & 0 \\ 011 & \longrightarrow & 0 \\ 010 & \longrightarrow & 0 \\ 001 & \longrightarrow & 1 \\ 000 & \longrightarrow & 0 \end{array}$$

The local rule replaces pattern 01 by pattern 10.

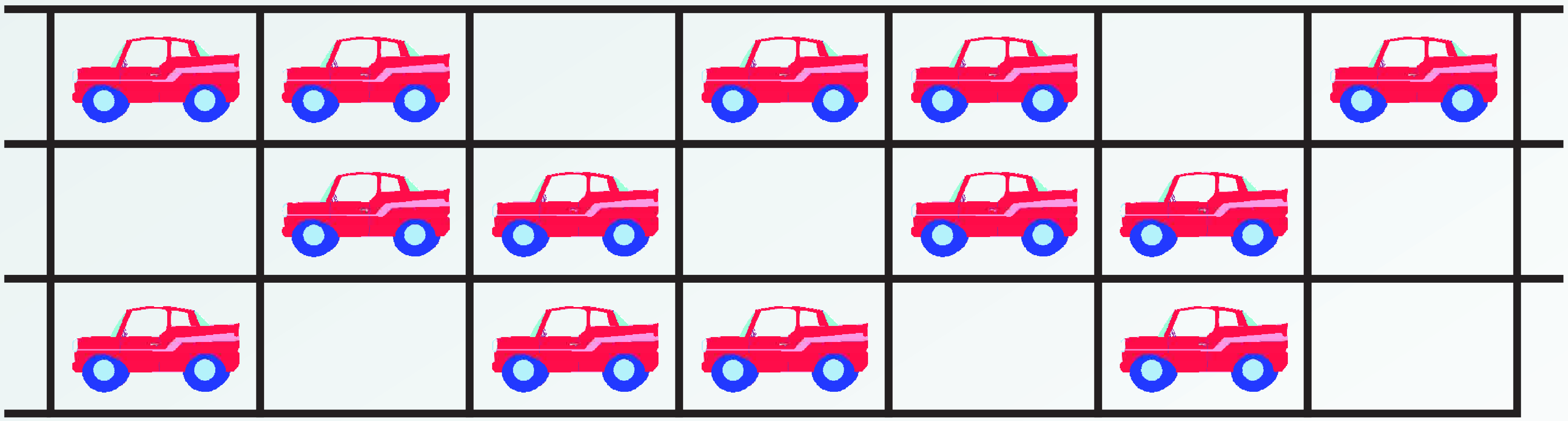
111	$\longrightarrow$	1
110	$\longrightarrow$	1
101	$\longrightarrow$	1
100	$\longrightarrow$	0
011	$\longrightarrow$	0
010	$\longrightarrow$	0
001	$\longrightarrow$	1
000	$\longrightarrow$	0



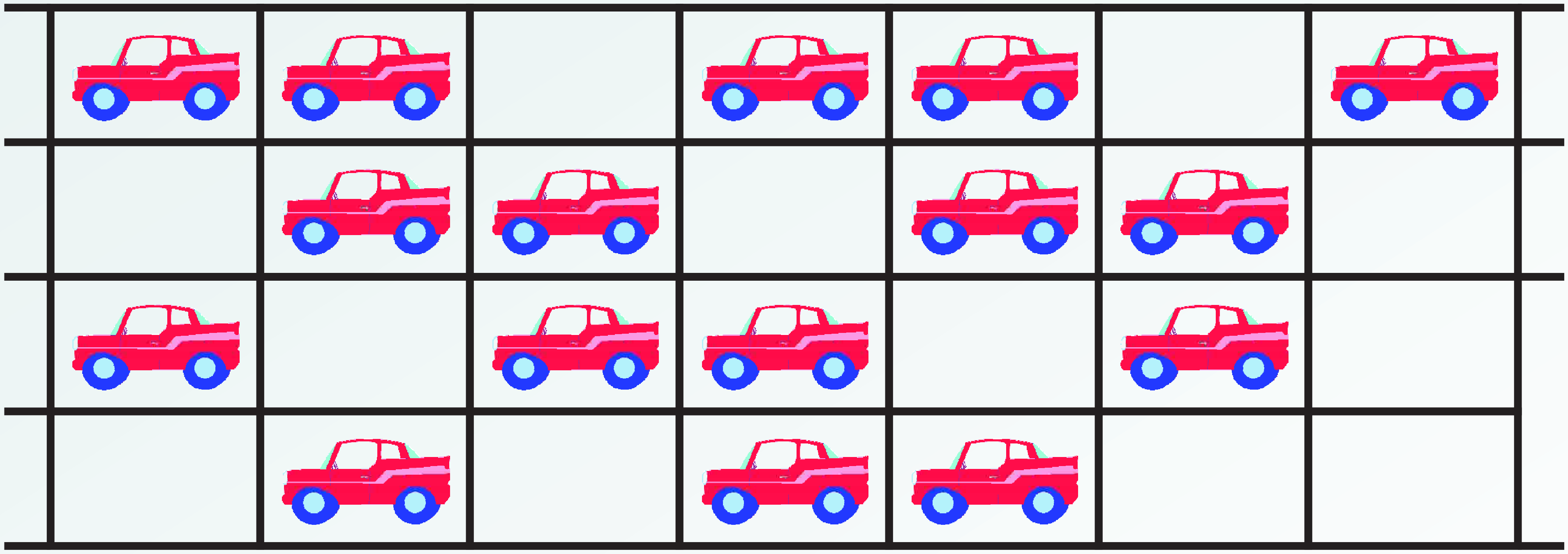
111	$\longrightarrow$	1
110	$\longrightarrow$	1
101	$\longrightarrow$	1
100	$\longrightarrow$	0
011	$\longrightarrow$	0
010	$\longrightarrow$	0
001	$\longrightarrow$	1
000	$\longrightarrow$	0



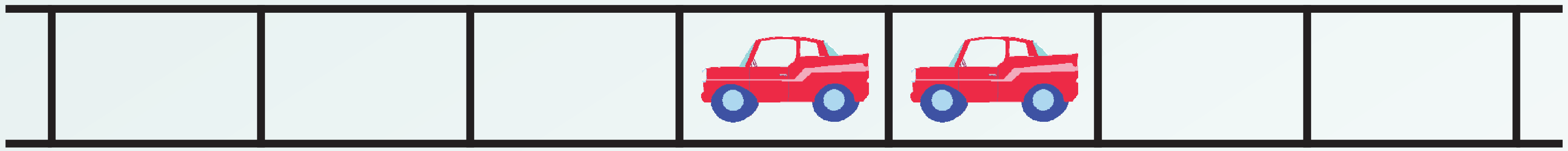
$$\begin{array}{ccc}
111 & \longrightarrow & 1 \\
110 & \longrightarrow & 1 \\
101 & \longrightarrow & 1 \\
100 & \longrightarrow & 0 \\
011 & \longrightarrow & 0 \\
010 & \longrightarrow & 0 \\
001 & \longrightarrow & 1 \\
000 & \longrightarrow & 0
\end{array}$$

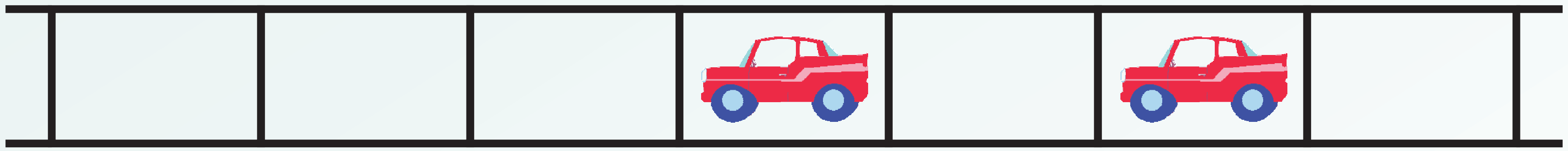


111	$\longrightarrow$	1
110	$\longrightarrow$	1
101	$\longrightarrow$	1
100	$\longrightarrow$	0
011	$\longrightarrow$	0
010	$\longrightarrow$	0
001	$\longrightarrow$	1
000	$\longrightarrow$	0



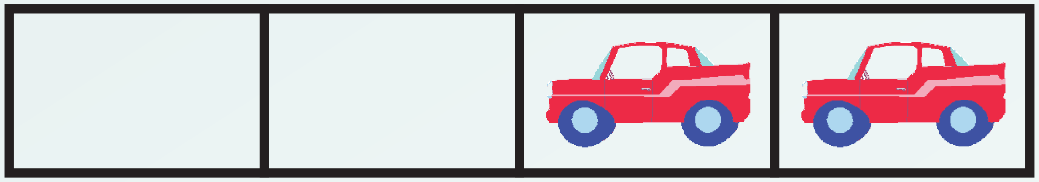
The local rule is balanced. However, there are two asymptotic configurations with the same successor:

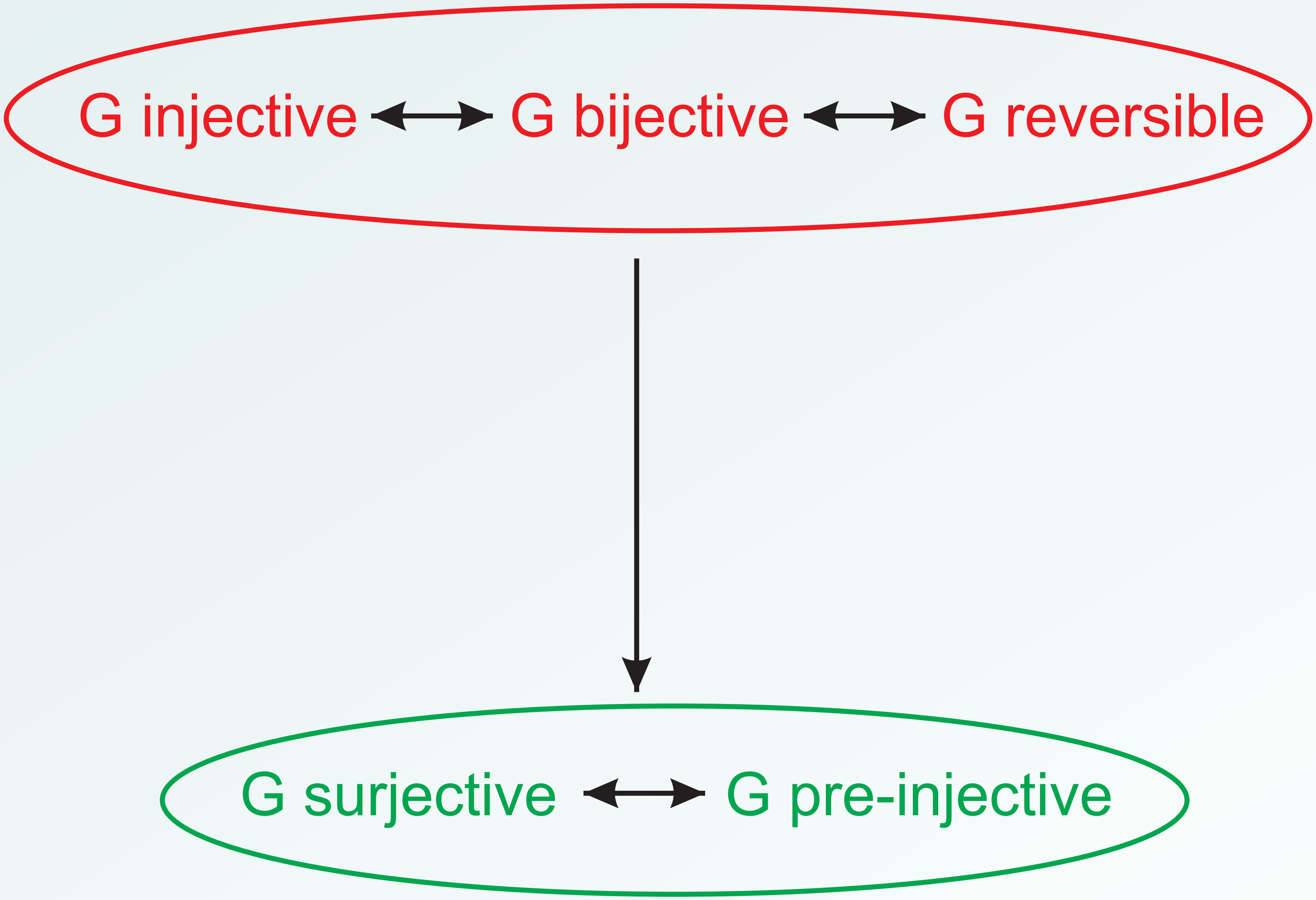


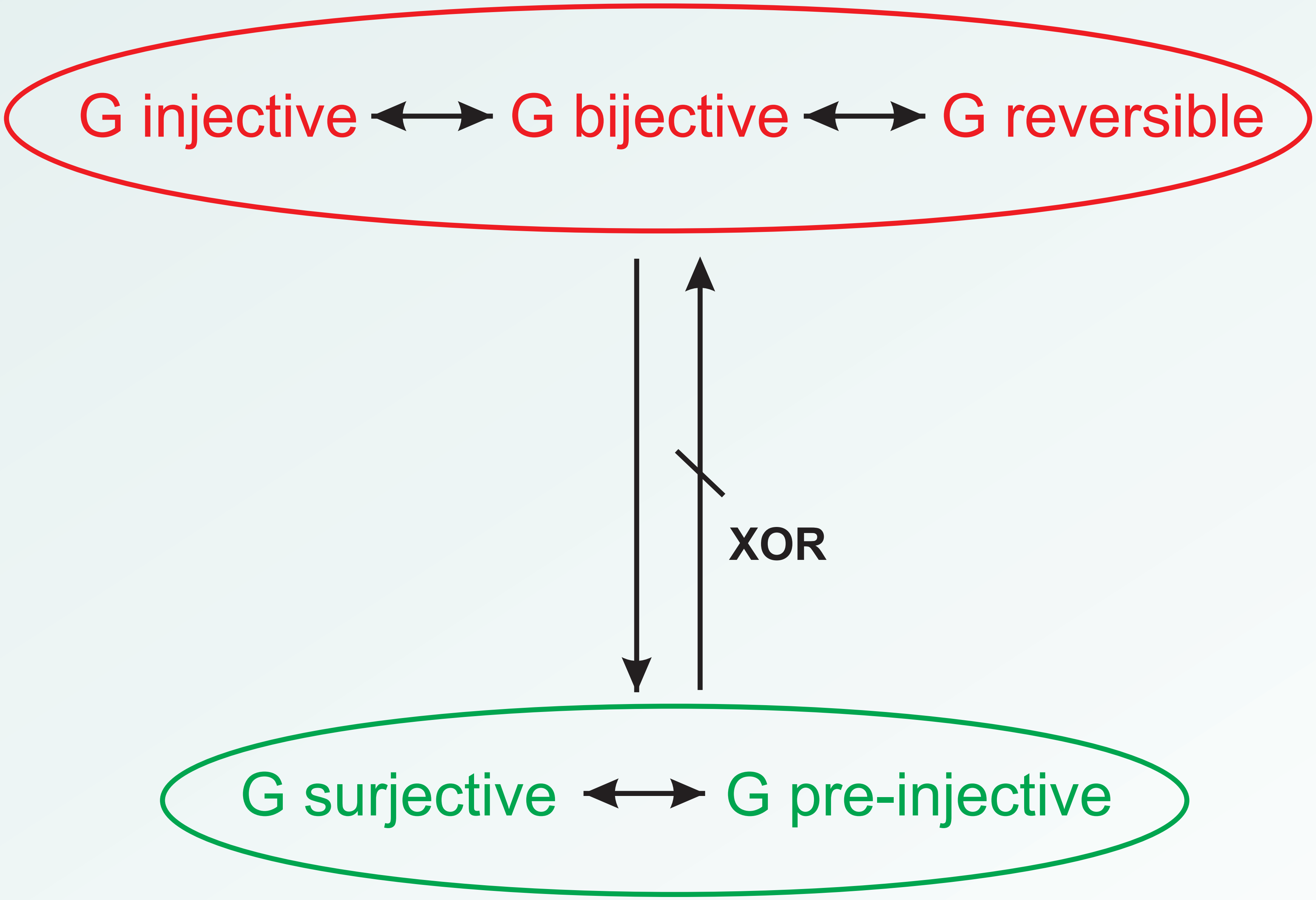


and hence traffic CA is not surjective.

There is an orphan of size four:




G injective
G bijective
G reversible
G surjective
G pre-injective


G injective
G bijective
G reversible
XOR
G surjective
G pre-injective

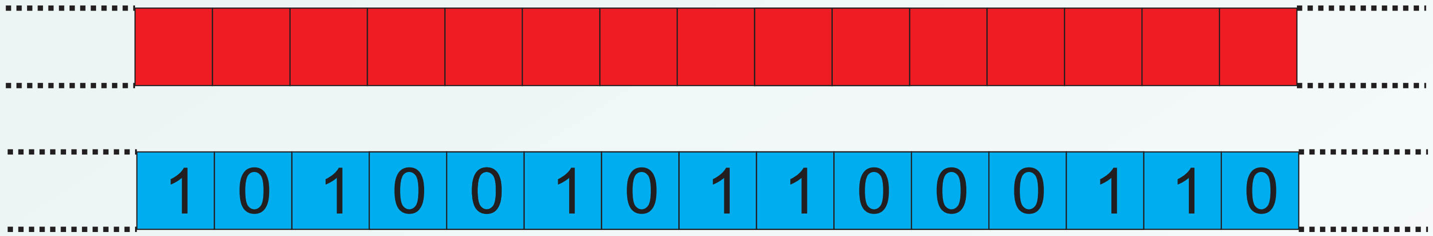
The xor-CA is the binary state CA with neighborhood $(0, 1)$ and local rule

$$f(a, b) = a + b \pmod 2.$$

The xor-CA is the binary state CA with neighborhood $(0, 1)$ and local rule

$$f(a, b) = a + b \pmod 2.$$

In the xor-CA every configuration has exactly two pre-images, so G is surjective but not injective:

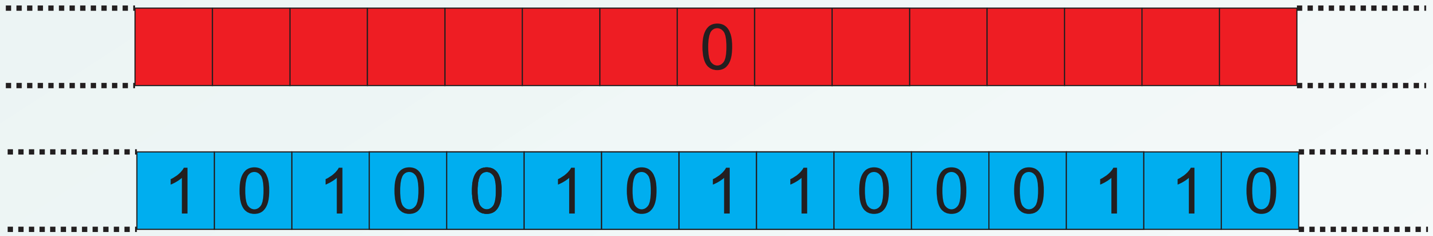


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

The xor-CA is the binary state CA with neighborhood $(0,1)$ and local rule

$$f(a,b) = a+b \pmod 2.$$

In the xor-CA every configuration has exactly two pre-images, so G is surjective but not injective:

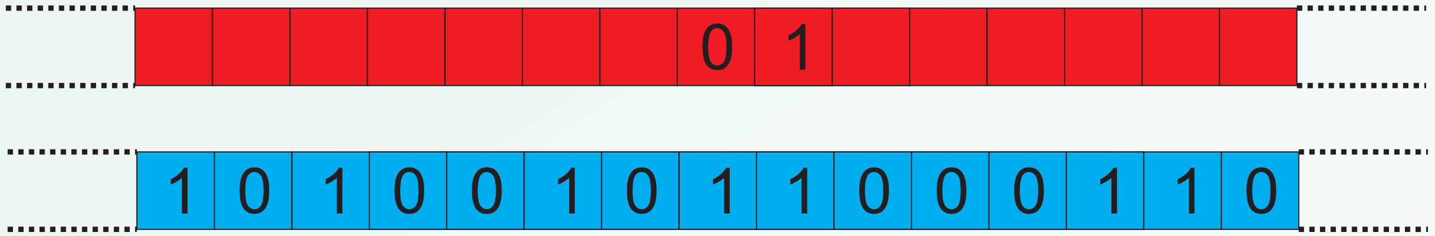


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

The xor-CA is the binary state CA with neighborhood $(0, 1)$ and local rule

$$f(a, b) = a + b \pmod{2}.$$

In the xor-CA every configuration has exactly two pre-images, so G is surjective but not injective:

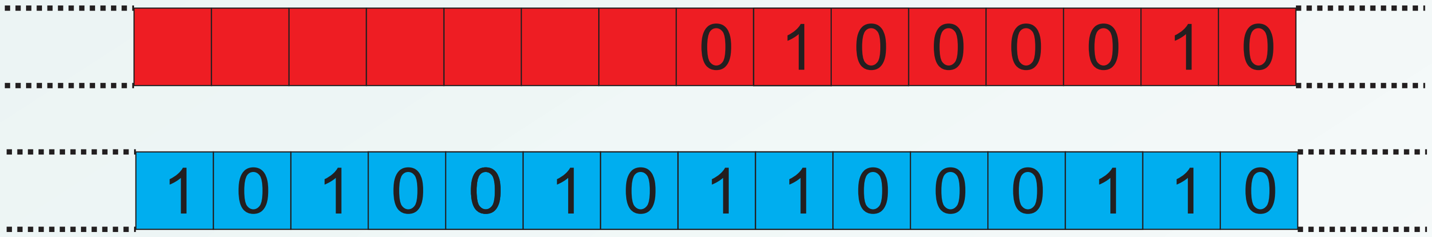


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

The xor-CA is the binary state CA with neighborhood $(0, 1)$ and local rule

$$f(a, b) = a + b \pmod 2.$$

In the xor-CA every configuration has exactly two pre-images, so G is surjective but not injective:

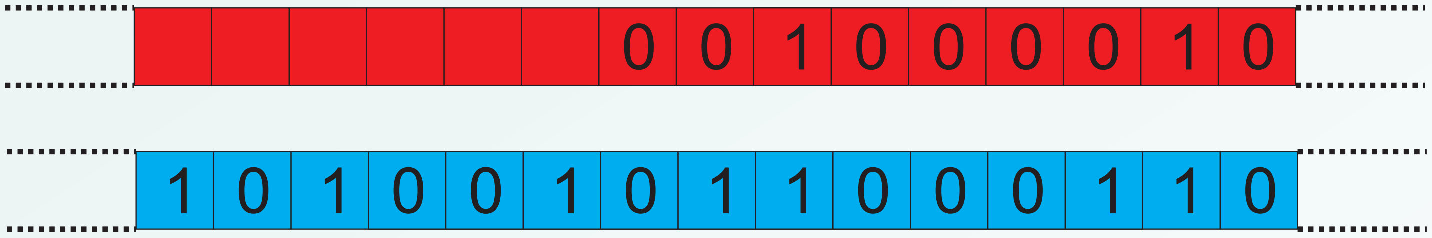


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

The xor-CA is the binary state CA with neighborhood $(0, 1)$ and local rule

$$f(a, b) = a + b \pmod 2.$$

In the xor-CA every configuration has exactly two pre-images, so G is surjective but not injective:

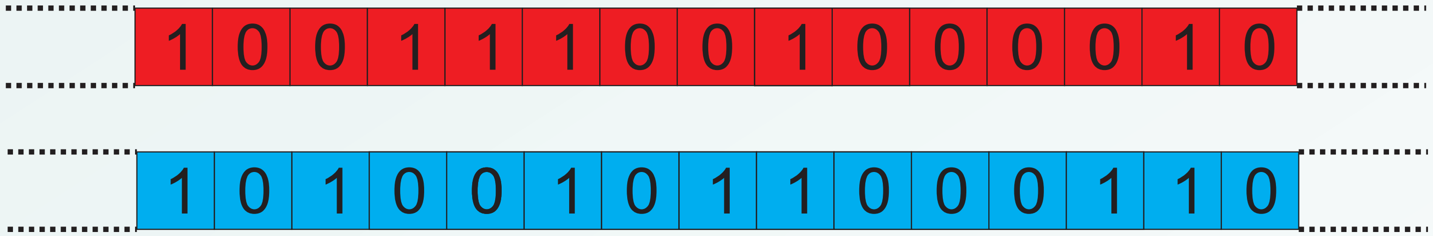


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

The xor-CA is the binary state CA with neighborhood $(0, 1)$ and local rule

$$f(a, b) = a + b \pmod 2.$$

In the xor-CA every configuration has exactly two pre-images, so G is surjective but not injective:



One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

Universality in reversible CA

Reversibility is an aspect of physics at the microscopic scale, so it is natural constraint to impose on a CA.

Simulating a Turing machine by an irreversible CA is trivial. Computation universality is, in fact, very common: Even simple one-dimensional rules (such as rule 110) can perform computations.

It turns out that computation universality is possible even under the reversibility constraint.

Theorem (T.Toffoli 1977). Any d-dimensional CA can be simulated by a $(d+1)$-dimensional RCA.

Corollary: Two-dimensional, computation universal RCA exist.

What about the one-dimensional case ?

What about the one-dimensional case ?

In 1989 K.Morita and M.Harao showed how to simulate any **reversible Turing machine** by a one-dimensional RCA. Since reversible Turing machines can be computation universal (Bennett 1973), the following result follows:

Theorem (K.Morita, M.Harao 1989). One-dimensional RCA exist that are computationally universal.

In 1995, J.-C. Dubacq showed how to simulate arbitrary (non-reversible) Turing machines in real time by a one-dimensional RCA.

The proof is based on Partitioned CA, another technique to guarantee reversibility.

The state set of a **partitioned CA** (PCA) is a cartesian product of finite sets:

$$S = S_1 \times S_2 \times \dots S_k.$$

The k components are **tracks**. The local update rule consists of two phases:

- a translation τ_i of each track $i = 1, 2, \dots, k$, and
- a bijection

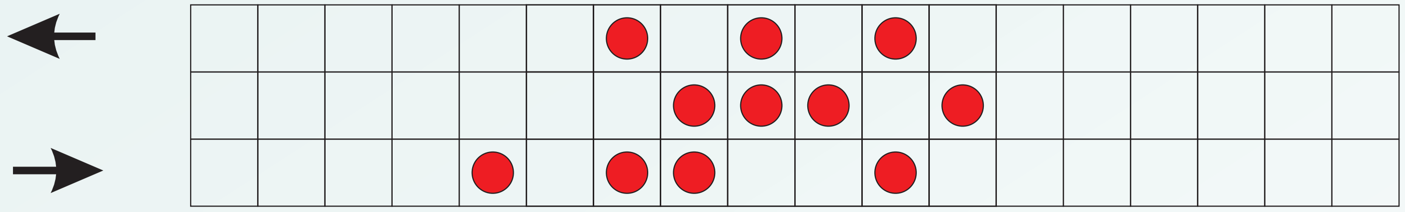
  $$\pi : S \longrightarrow S$$

  applied in each cell independent of its neighbors.

The two phases alternate.

Example: PCA with three binary tracks:

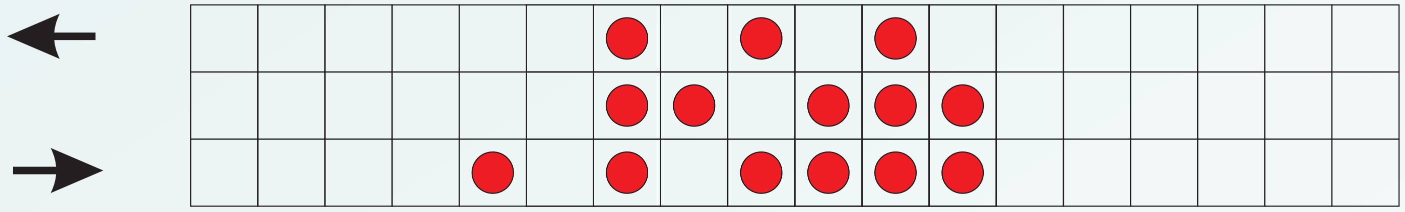
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \mapsto \quad \begin{bmatrix} a & \\ a+b & \pmod 2 \\ b+c & \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

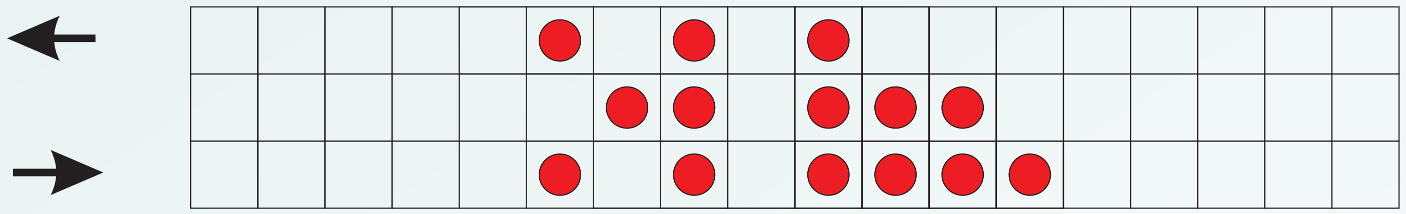
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \mapsto \begin{bmatrix} a & \\ a+b & \pmod 2 \\ b+c & \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

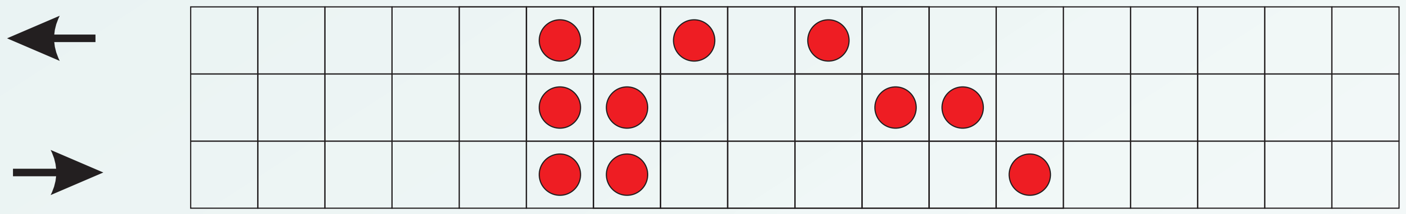
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \mapsto \quad \begin{bmatrix} a & \\ a+b & \pmod 2 \\ b+c & \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

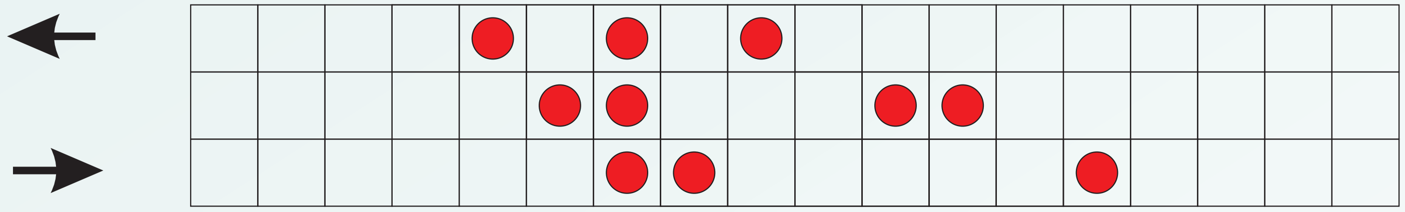
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \mapsto \quad \begin{bmatrix} a & \\ a+b & \pmod 2 \\ b+c & \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

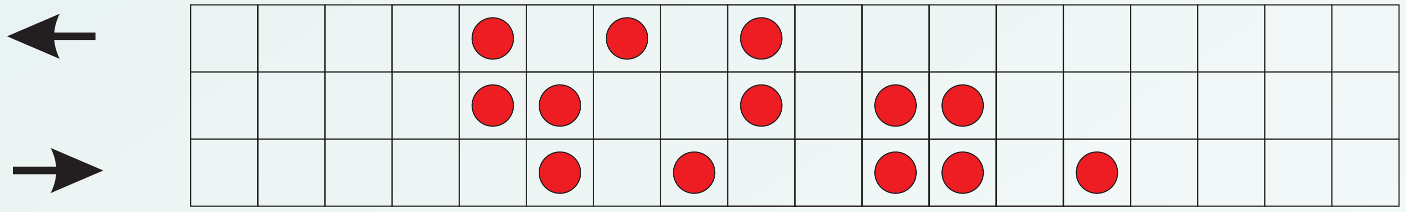
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \mapsto \begin{bmatrix} a \\ a+b \pmod 2 \\ b+c \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

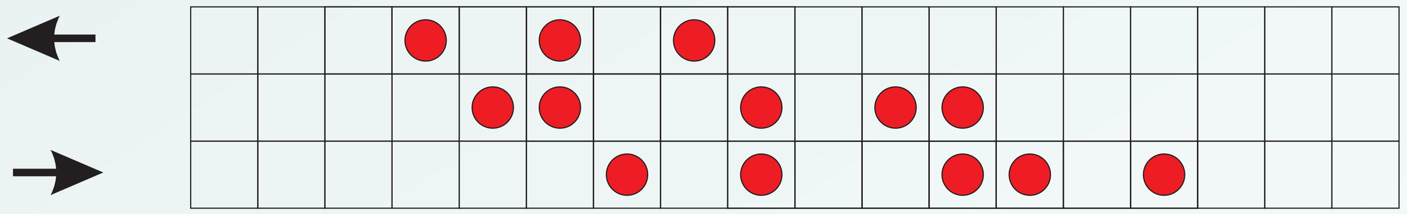
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \mapsto \begin{bmatrix} a \\ a+b \pmod 2 \\ b+c \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \mapsto \begin{bmatrix} a & \\ a+b & \pmod 2 \\ b+c & \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

To simulate a Turing machine we use a PCA with four tracks:

Track (1) the tape of the Turing machine. It is not translated.

To simulate a Turing machine we use a PCA with four tracks:

Track (1) the tape of the Turing machine. It is not translated.

Track (2) or (3) stores the Turing machine state. They are shifted one cell left and right respectively. The state is stored on the track which moves to the direction indicated by the TM instruction being executed.

To simulate a Turing machine we use a PCA with four tracks:

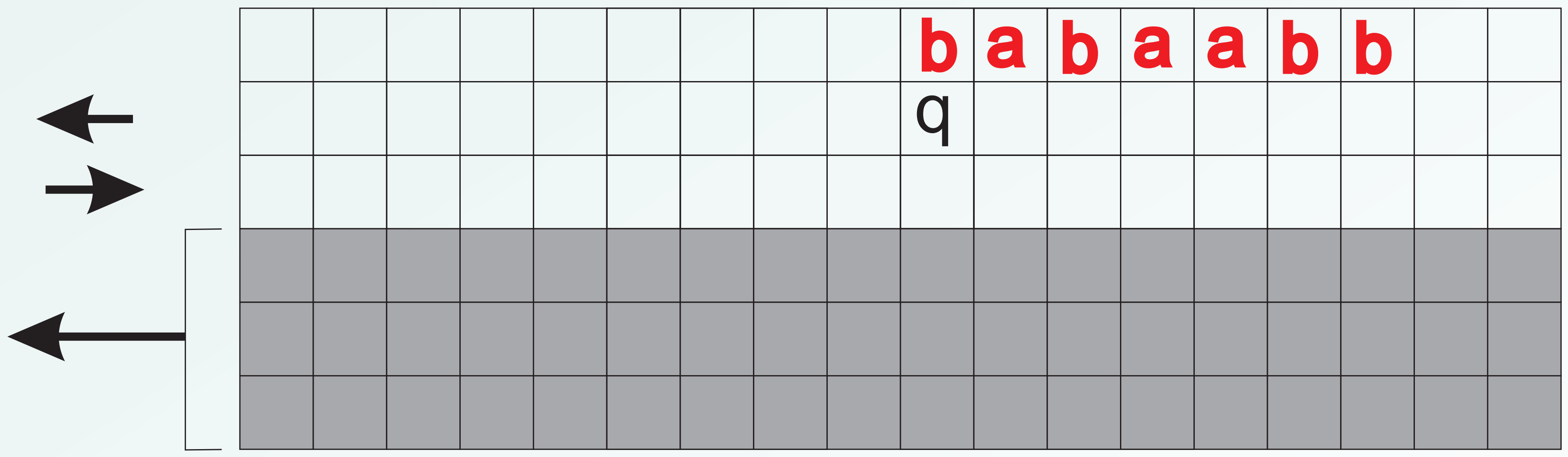
Track (1) the tape of the Turing machine. It is not translated.

Track (2) or (3) stores the Turing machine state. They are shifted one cell left and right respectively. The state is stored on the track which moves to the direction indicated by the TM instruction being executed.

Track (4) is a garbage track. It is translated by two cells so that a new empty ”trash bin” always appears at the position of the Turing machine.

The permutation π

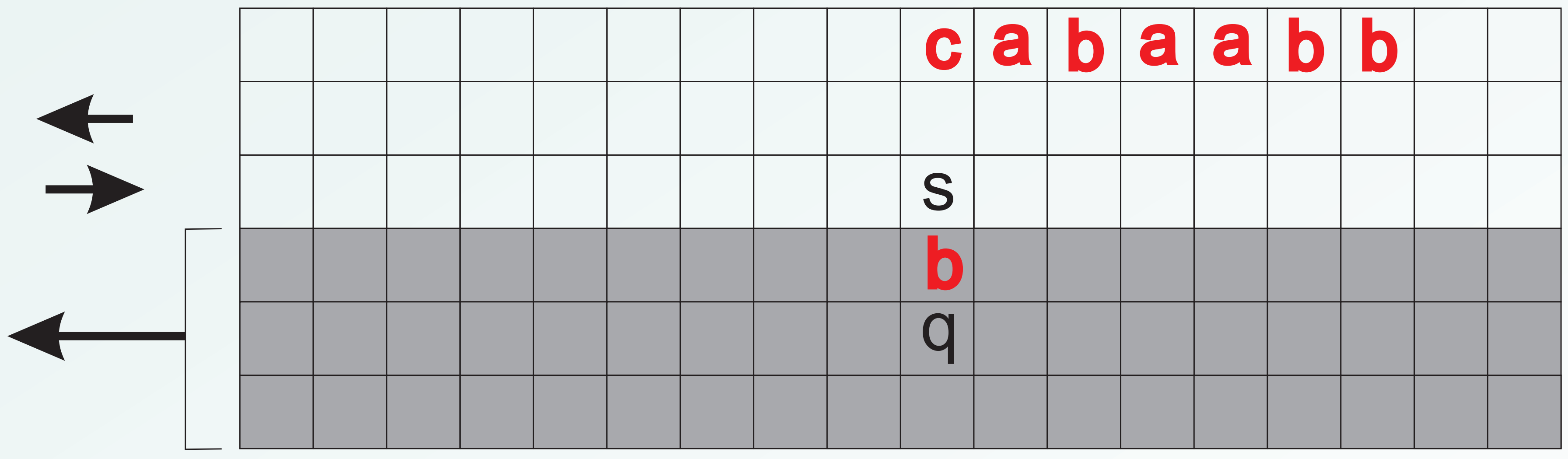
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



TM(q,b) = (s,c, $\rightarrow$)

The permutation π

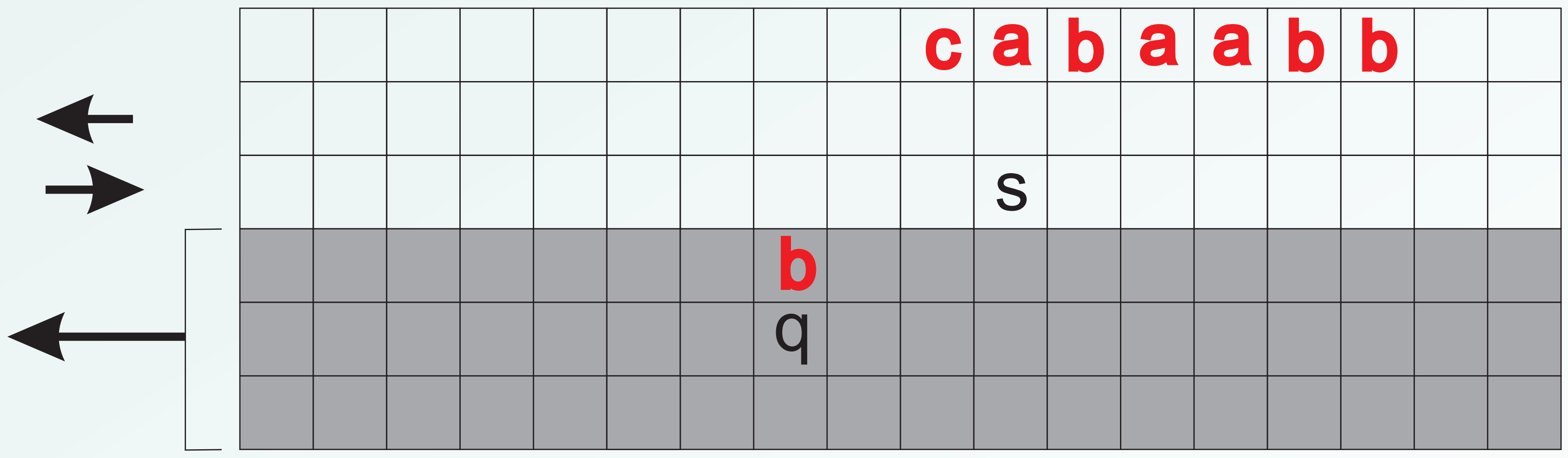
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



TM(q,b) = (s,c, $\rightarrow$)

The permutation π

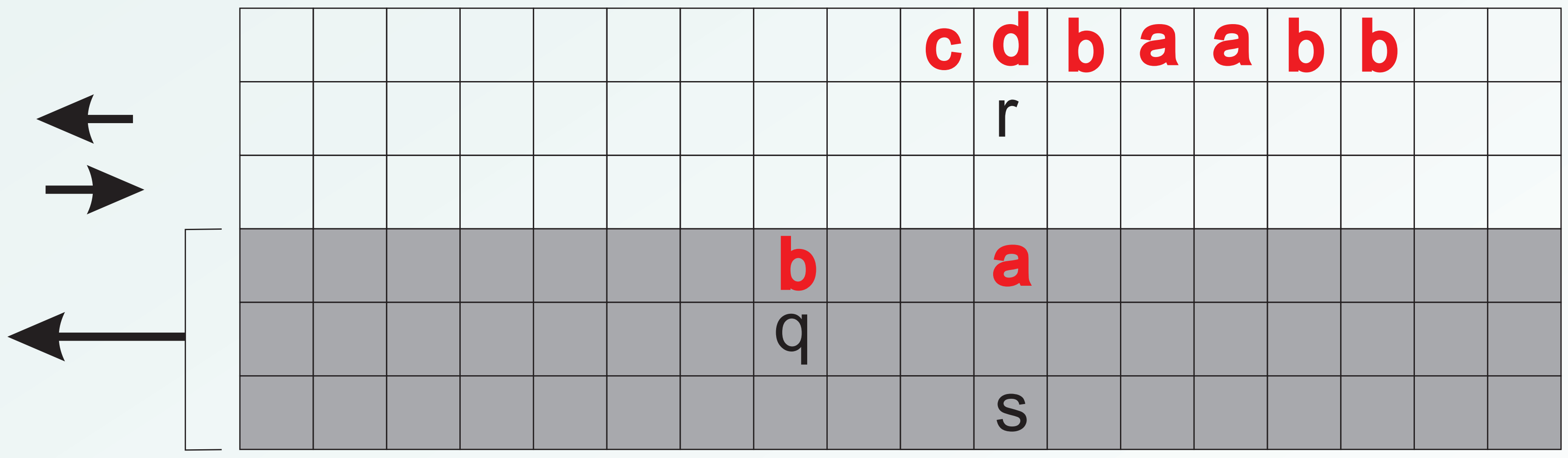
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



TM(s,a) = (r,d, $\leftarrow$)

The permutation π

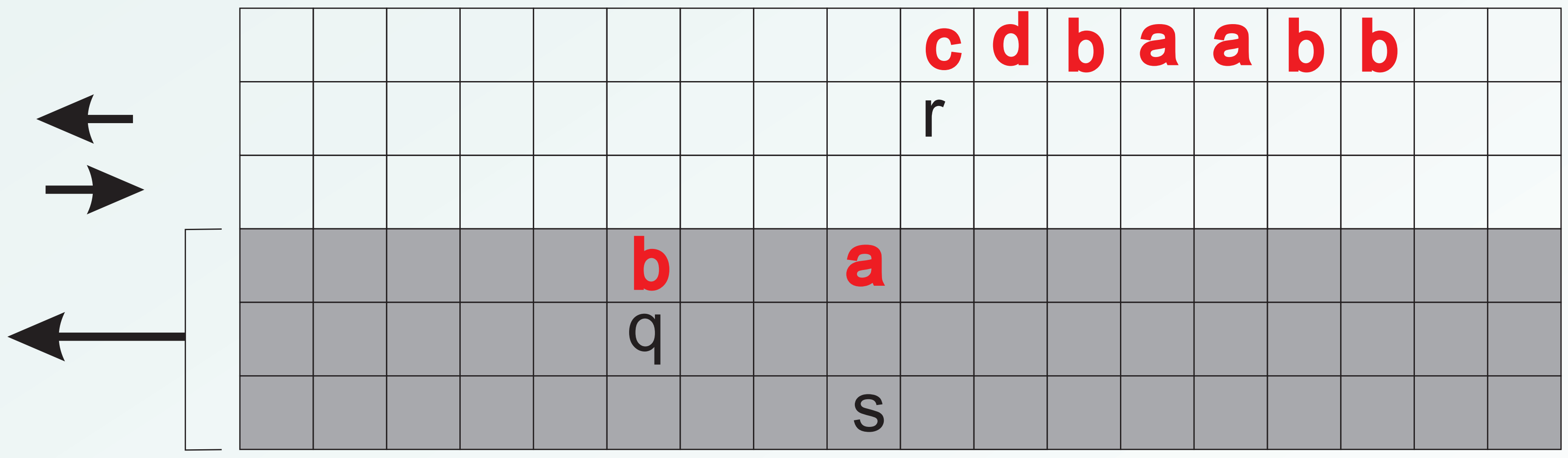
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



TM(s,a) = (r,d, $\leftarrow$)

The permutation π

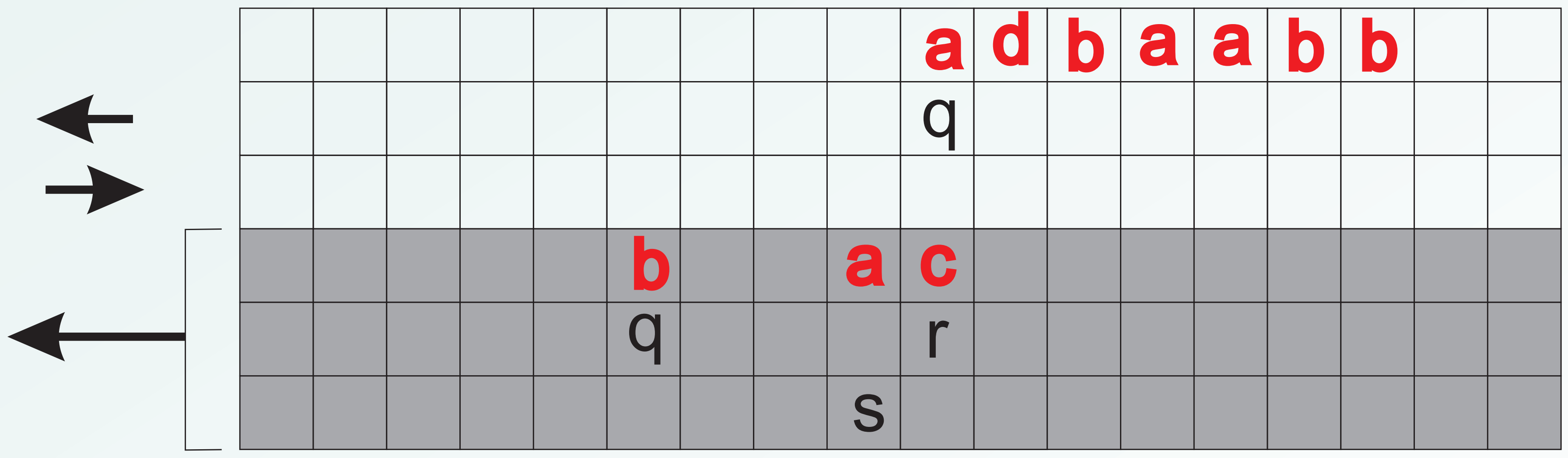
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



$\mathrm{TM}(r,c) = (q,a,\leftarrow)$

The permutation π

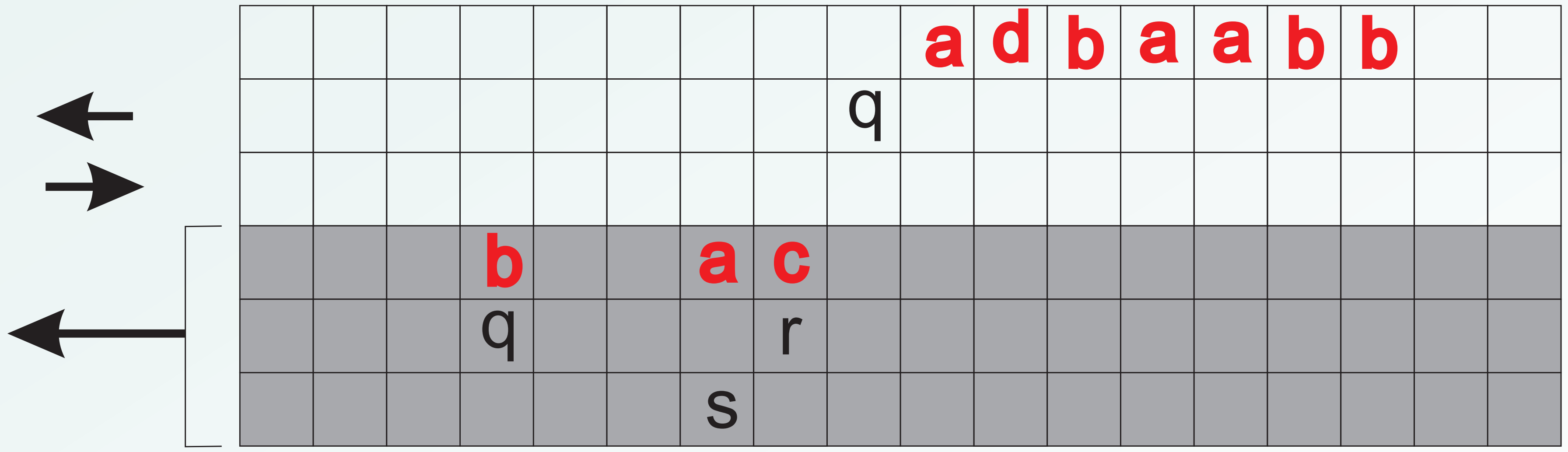
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



$$\mathrm{TM}(r,c) = (q,a,\leftarrow)$$

The permutation π

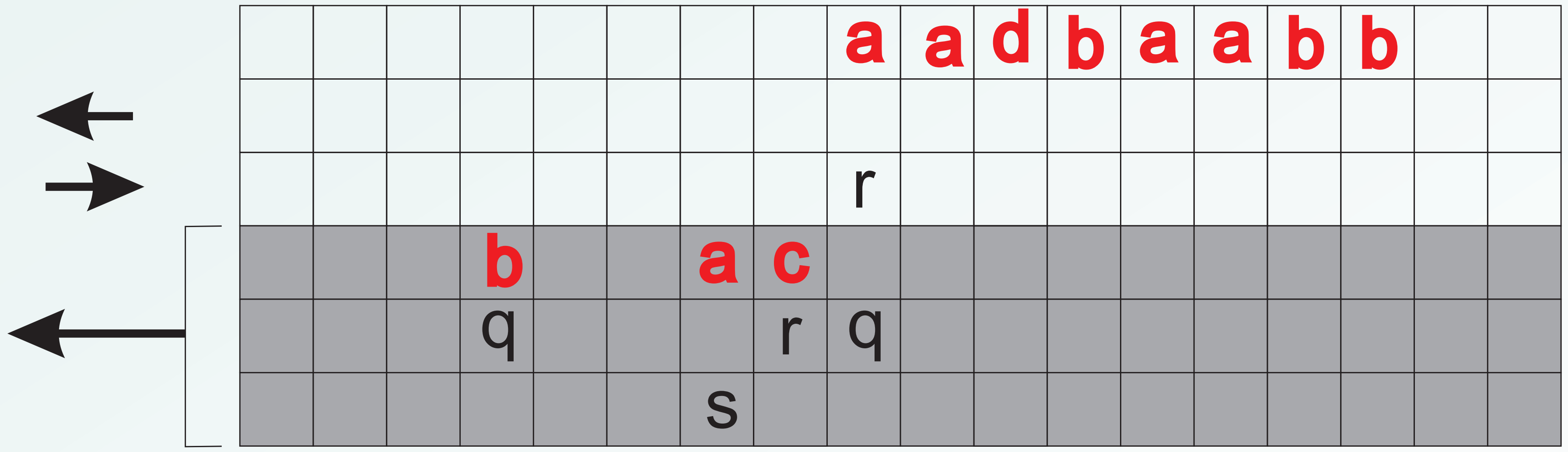
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



TM(q,) = (r,a,$\rightarrow$)

The permutation π

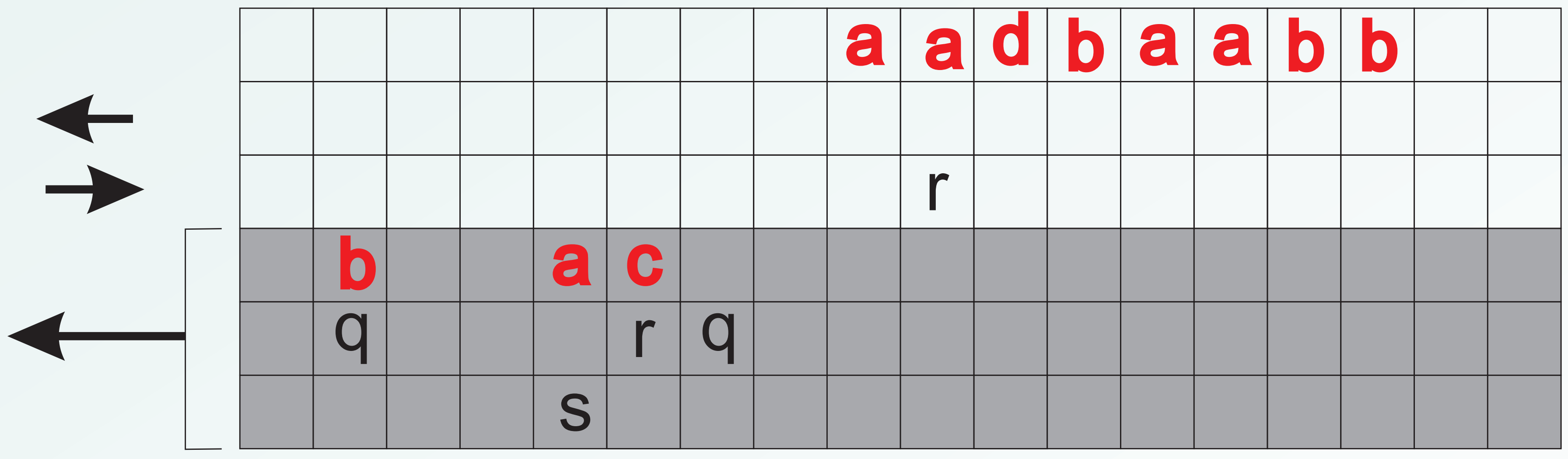
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



TM(q,   ) = (r,a,$\rightarrow$)

The permutation π

- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



TM(r, a) = (s,d, $\rightarrow$)

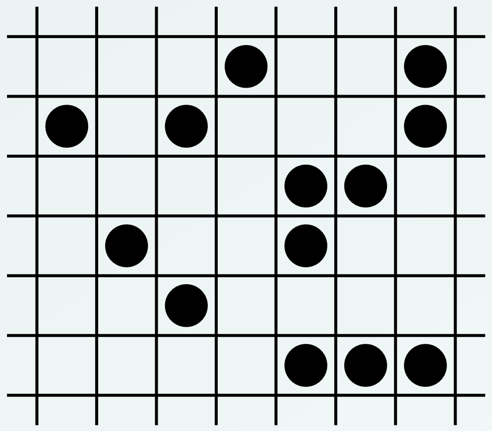
The **partially** defined π is one-to-one.

Any partially defined one-to-one map $S \longrightarrow S$ can be completed into a bijection by matching the missing elements in the domain and range **arbitrarily**.

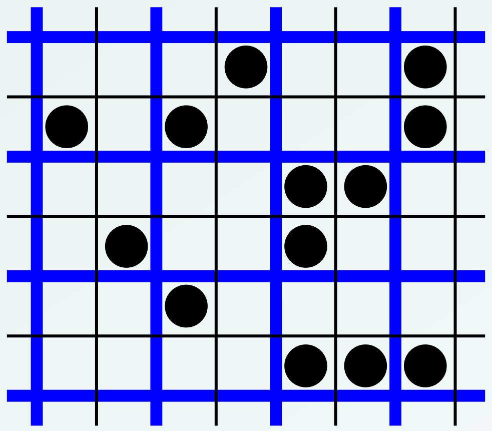
This completes the construction. $\square$

A particular universal two-dimensional RCA was designed by N.Margolus, based on the **Billiard Ball Model** of computation. This RCA also introduced the technique of **block permutations** to guarantee reversibility.

In the block permutation technique the updating is done in two steps:

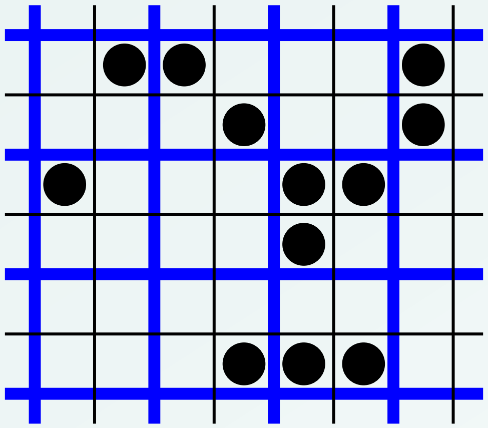


In the block permutation technique the updating is done in two steps:



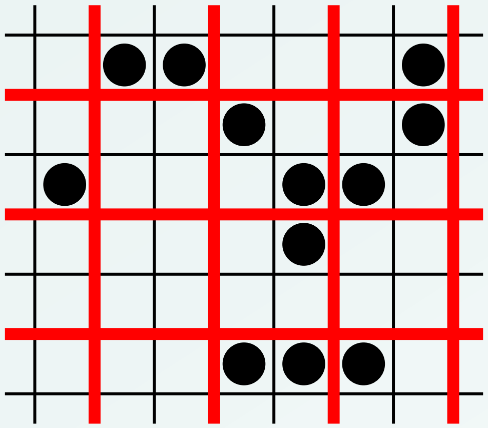
1. Partition the plane into 2×2 blocks and apply some permutation π_1 of S^4 inside each block.

In the block permutation technique the updating is done in two steps:



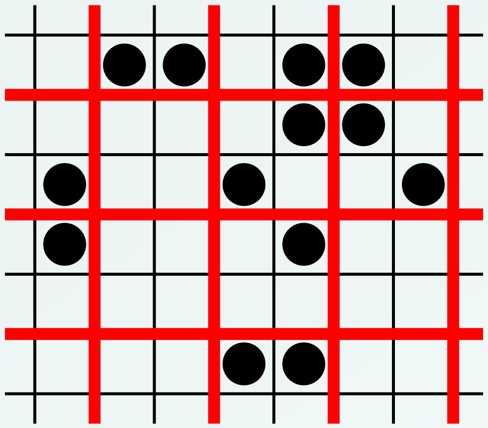
1. Partition the plane into 2×2 blocks and apply some permutation π_1 of S^4 inside each block.

In the block permutation technique the updating is done in two steps:



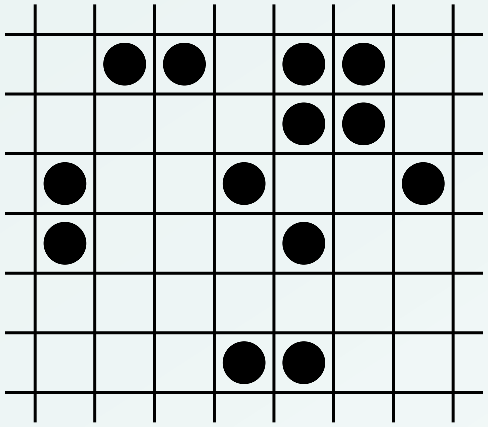
2. Shift the partitioning horizontally and vertically, and apply another permutation π_2 of S^4 on the new blocks.

In the block permutation technique the updating is done in two steps:



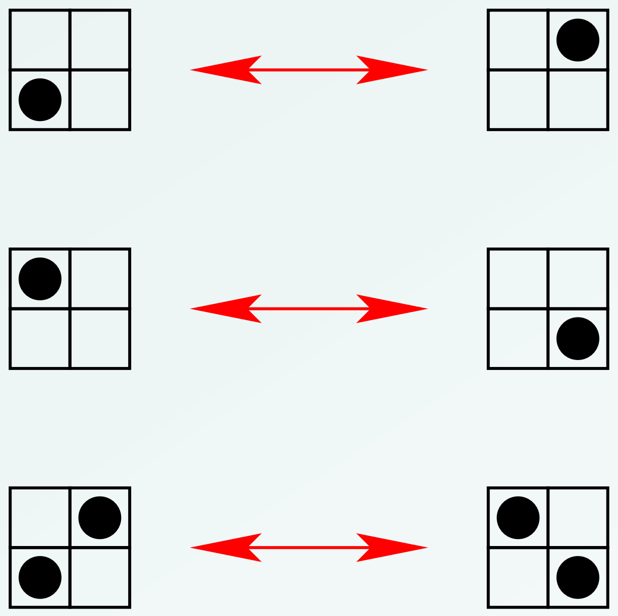
2. Shift the partitioning horizontally and vertically, and apply another permutation π_2 of S^4 on the new blocks.

In the block permutation technique the updating is done in two steps:

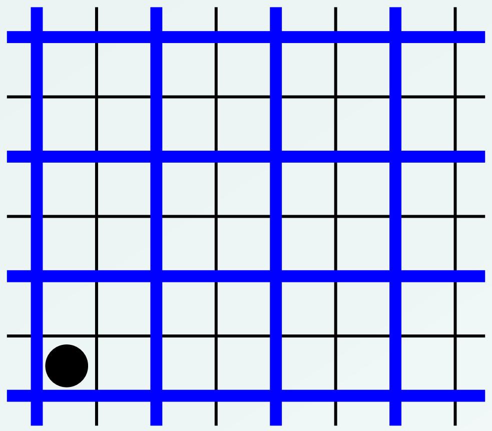


The composition of the two block permutation is one iteration of the CA. It is trivially reversible.

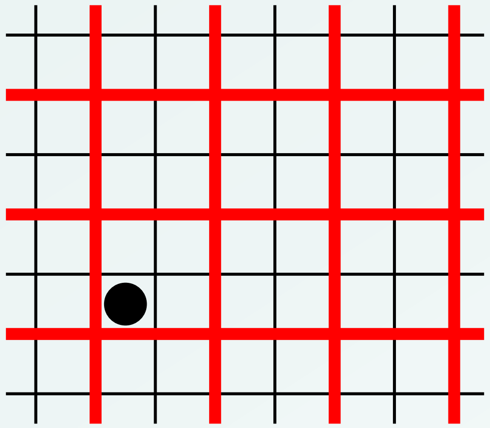
In the Billiard Ball RCA S is a binary set, and both permutations π_1 and π_2 are identical. They only make the following exchanges:



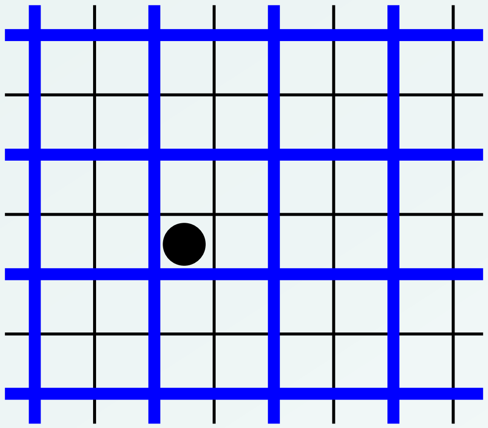
Because of alternating partitioning, a single black state propagates in one of the four diagonal directions:



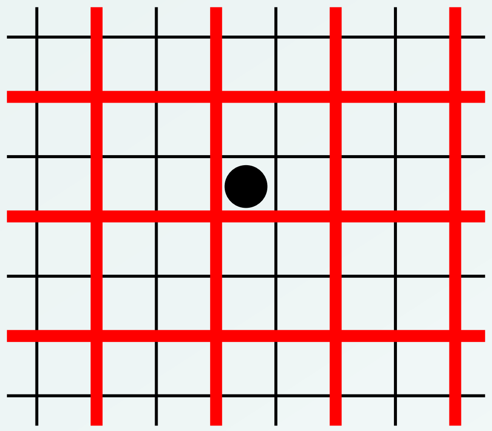
Because of alternating partitioning, a single black state propagates in one of the four diagonal directions:



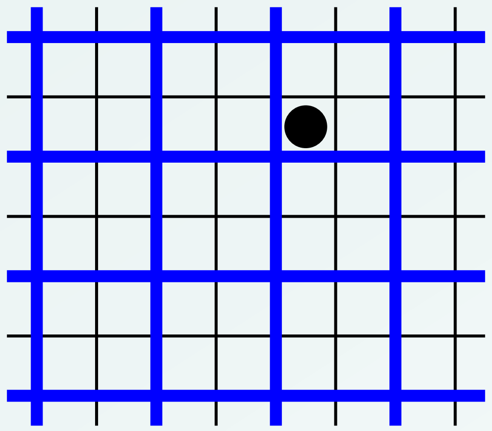
Because of alternating partitioning, a single black state propagates in one of the four diagonal directions:



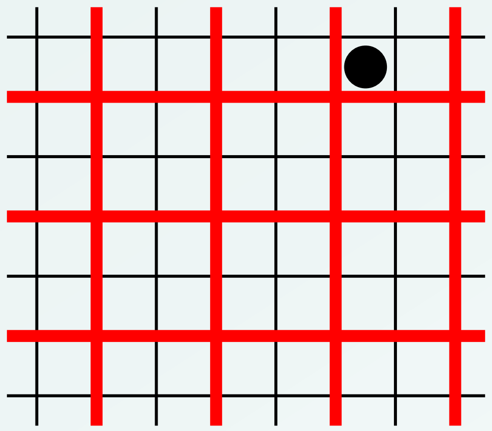
Because of alternating partitioning, a single black state propagates in one of the four diagonal directions:



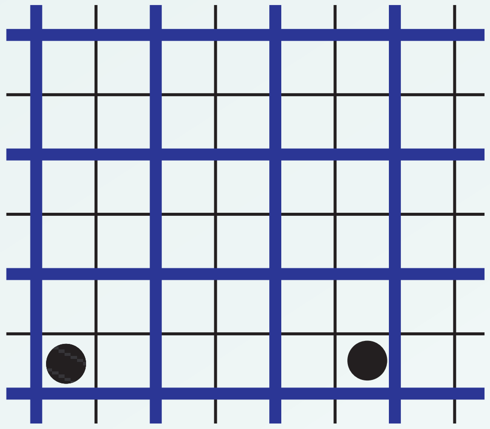
Because of alternating partitioning, a single black state propagates in one of the four diagonal directions:



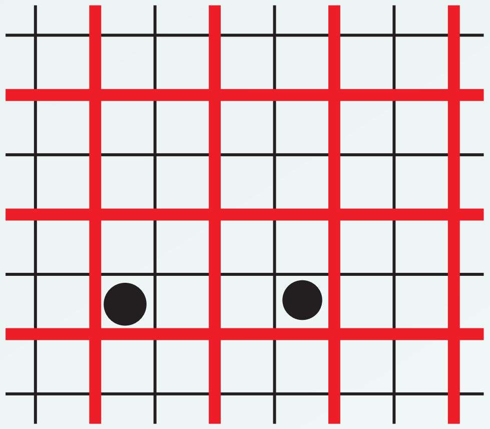
Because of alternating partitioning, a single black state propagates in one of the four diagonal directions:



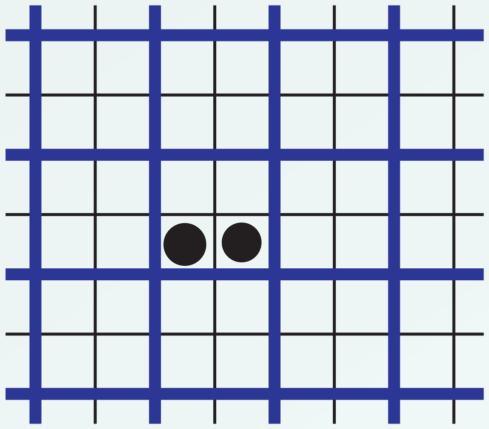
Two particles that collide sideways pause for a moment and reverse their direction:



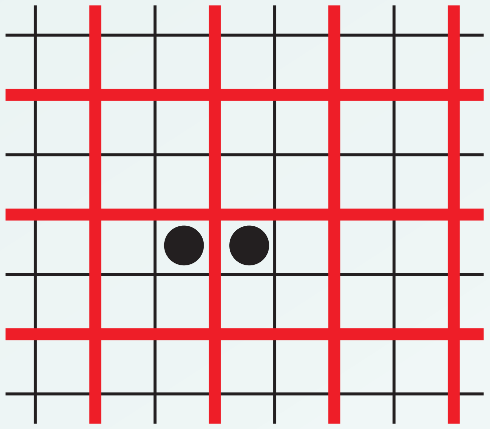
Two particles that collide sideways pause for a moment and reverse their direction:



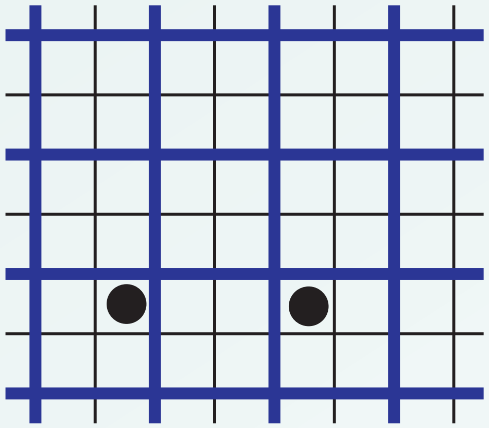
Two particles that collide sideways pause for a moment and reverse their direction:



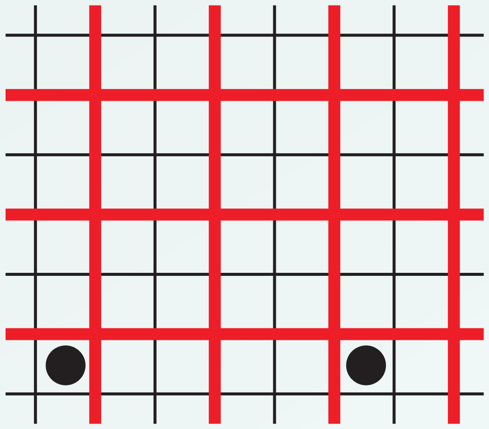
Two particles that collide sideways pause for a moment and reverse their direction:



Two particles that collide sideways pause for a moment and reverse their direction:



Two particles that collide sideways pause for a moment and reverse their direction:



In the Billiard Ball RCA one can simulate the motion and collisions of balls of positive diameter. Walls from which the balls bounce can also be created. These constructs are sufficient to implement universal reversible gates.

In the Billiard Ball RCA one can simulate the motion and collisions of balls of positive diameter. Walls from which the balls bounce can also be created. These constructs are sufficient to implement universal reversible gates.

Reversibility is trivially obtained using the block permutation technique. Also **conservation laws** are easy to enforce.

For example, the Billiard Ball RCA conserves the total number of black states.

local reversibility vs. global reversibility

Reversible CA come up naturally when simulating reversible physical systems.

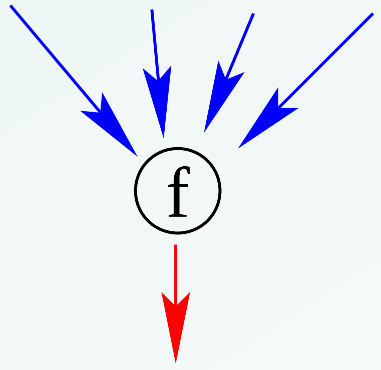
local reversibility vs. global reversibility

Reversible CA come up naturally when simulating reversible physical systems.

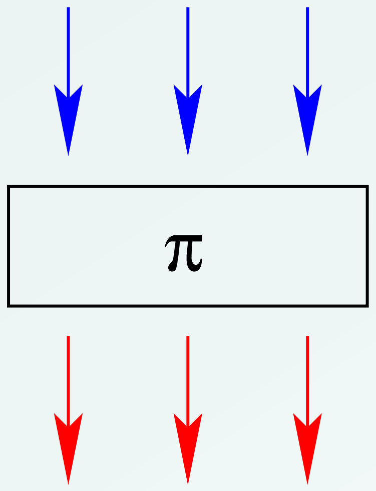
Conversely, physically building a cellular automata device in our reversible physical universe is most **energy efficient** if the CA is implemented using reversible gates.

A problem with reversible CA is that reversibility is **global**: the transformation $G : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$ is bijective on **infinite configurations**.

In order to exploit the reversibility of RCA they should be implemented using **finite reversible logical gates**. However, CA are given in terms of the local update rule $f : S^n \longrightarrow S$ that is **not** reversible:

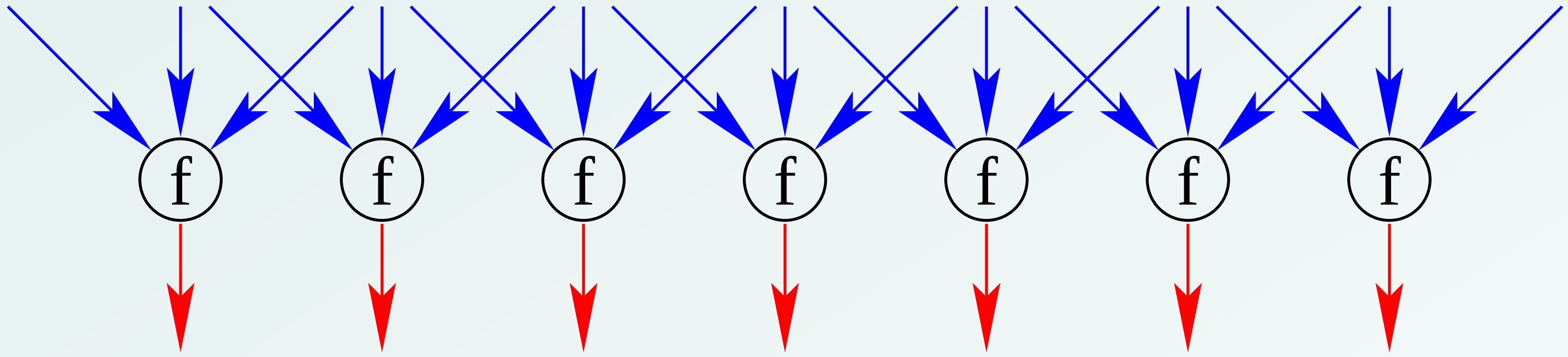


Rather than using the non-reversible local rule f we would need to use reversible local functions

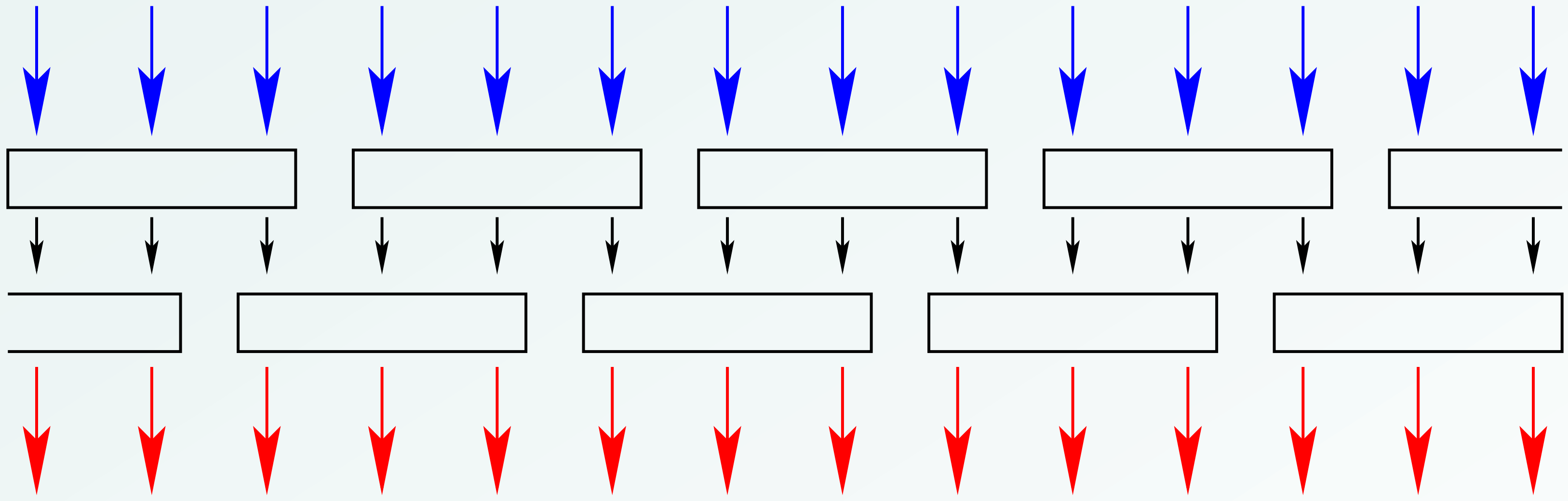


Partitioned CA, and the block partitioning technique of the Billiard Ball RCA are examples where the local rule was given in terms of reversible local rules. A natural question: Can every RCA be expressed using reversible local rules ?

In other words, we would like to replace a traditional CA



with something like



that computes the same global function $G : S^{\mathbb{Z}^d} \longrightarrow S^{\mathbb{Z}^d}$.

The answer is **positive** in one- and two-dimensional spaces:

• Every one-dimensional RCA can be built using two levels of block permutations (and some translations).

• Every two-dimensional RCA can be built using three levels of block permutations (and some translations).

The answer is **positive** in one- and two-dimensional spaces:

• Every one-dimensional RCA can be built using two levels of block permutations (and some translations).

• Every two-dimensional RCA can be built using three levels of block permutations (and some translations).

The question is **open** for three-dimensional CA.

Conjecture: Every d-dimensional RCA can be expressed as $d + 1$ layers of block permutations and translations.

Periodic configurations

It is obviously not possible to simulate CA functions on arbitrary infinite configurations, but one has to limit the attention to some subset of $S^{\mathbb{Z}^d}$.

We often consider the action on **periodic configurations**.

Periodic configurations: Configuration $c \in S^{\mathbb{Z}^d}$ has **period** $\vec{r} \in \mathbb{Z}^d$ if it is invariant under the translation τ by $\vec{r}$:

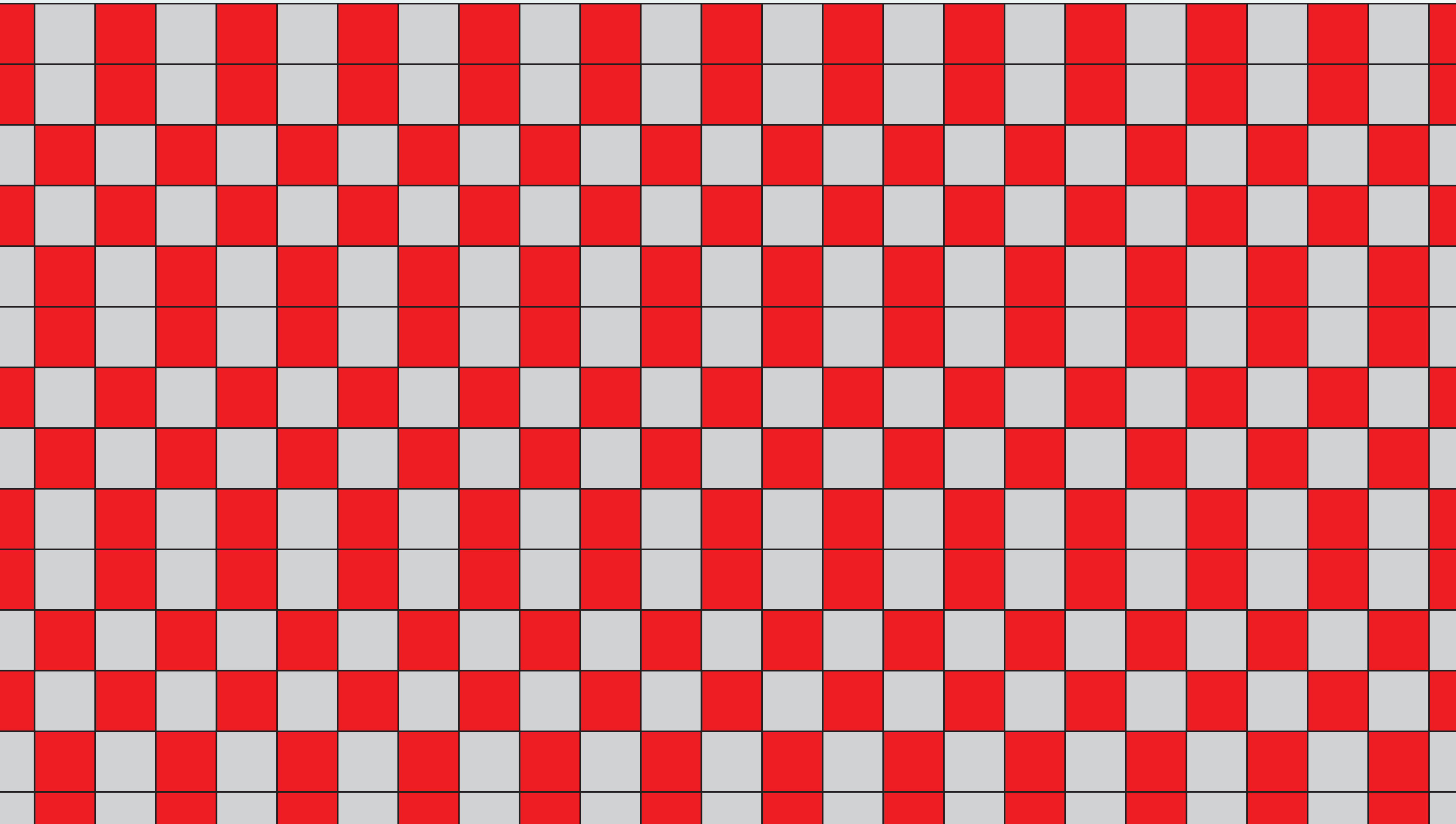
$$\tau(c) = c.$$

CA functions commute with translations, so we also have

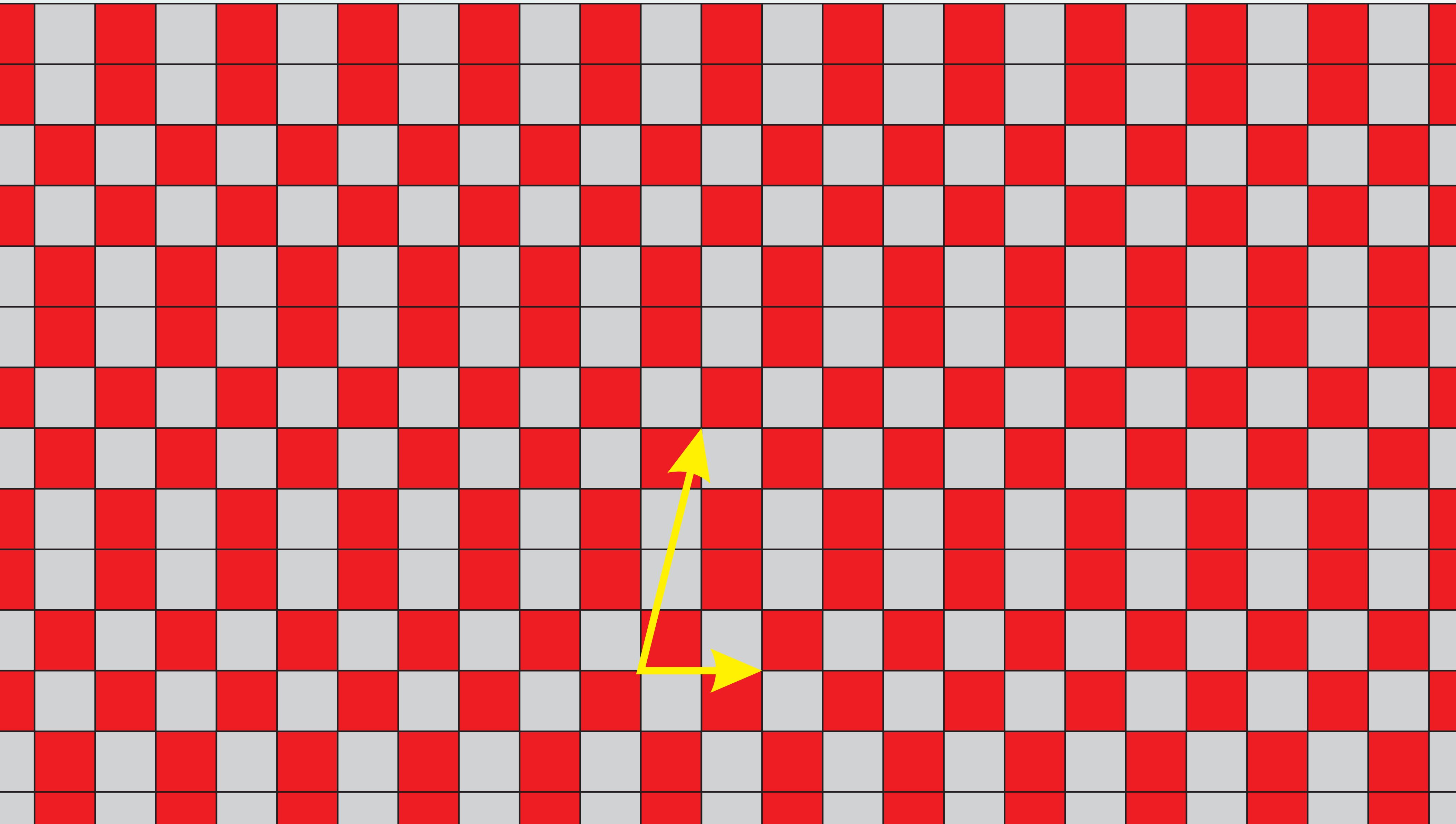
$$\tau(G(c)) = G(\tau(c)) = G(c).$$

Period $\vec{r}$ of c is also a period of $G(c)$.

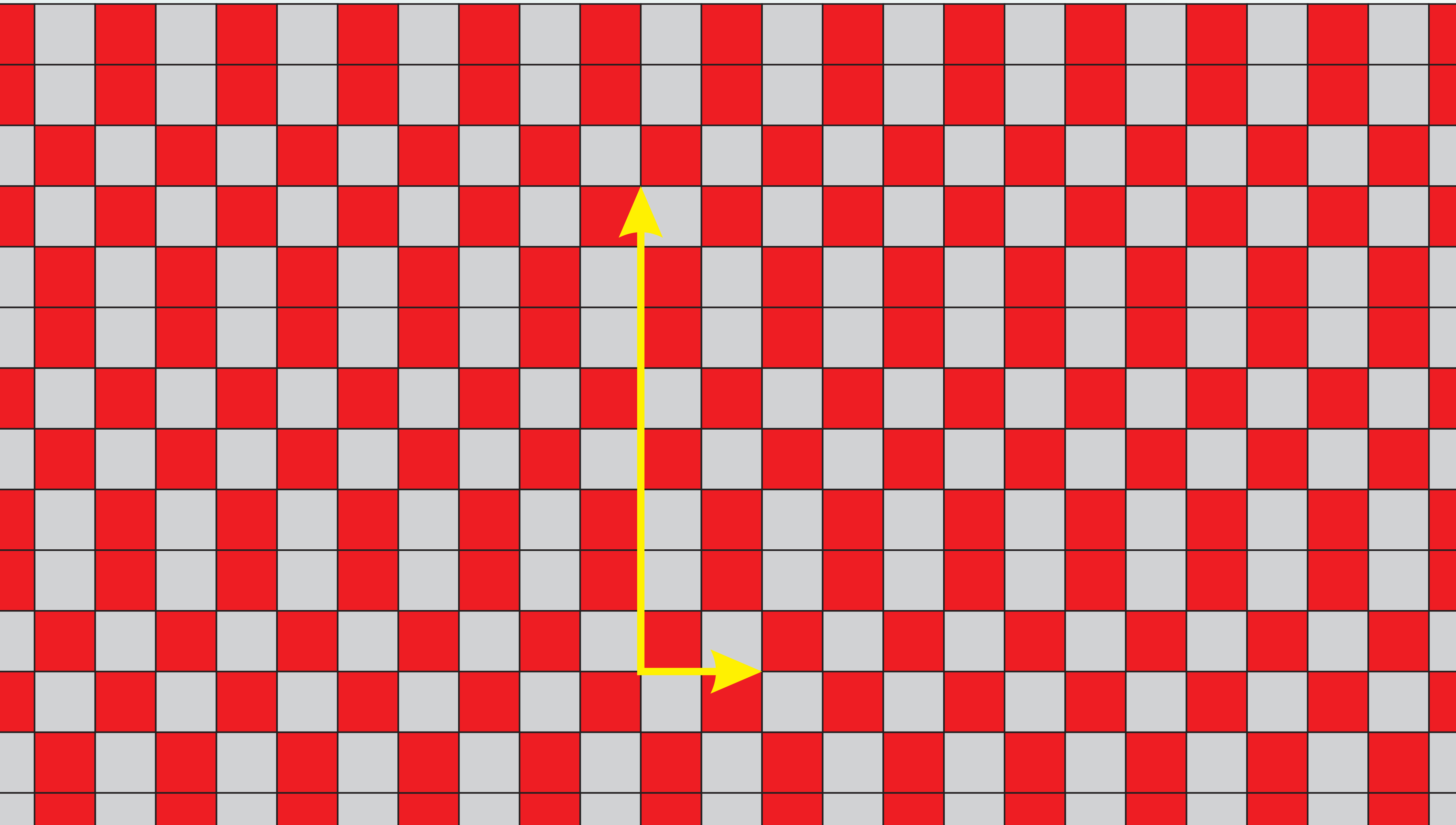
Configuration $c \in S^{\mathbb{Z}^d}$ is (fully) **periodic** if it has d linearly independent periods.



Configuration $c \in S^{\mathbb{Z}^d}$ is (fully) **periodic** if it has d linearly independent periods.



Configuration $c \in S^{\mathbb{Z}^d}$ is (fully) **periodic** if it has d linearly independent periods.



Cellular automata preserve periods, so periodic configurations are mapped to periodic configurations.

Cellular automata preserve periods, so periodic configurations are mapped to periodic configurations.

The use of periodic configurations is usually termed **periodic boundary conditions**.

Periodic configurations are **dense** in the metric space $S^{\mathbb{Z}^d}$.

Surjectivity and injectivity of G_P

Let G_P denote the restriction of cellular automaton G on (fully) periodic configurations.

Implications

$$\begin{array}{rcl} \mathbf{G}\text{ injective} & \Longrightarrow & \mathbf{G_P}\text{ injective} \\ \mathbf{G_P}\text{ surjective} & \Longrightarrow & \mathbf{G}\text{ surjective} \end{array}$$

are easy. (Second one uses denseness of periodic configurations in $S^{\mathbb{Z}^d}$.)

We also have

$$\mathbf{G}_{\mathbf{P}} \text{ injective } \implies \mathbf{G}_{\mathbf{P}} \text{ surjective}$$

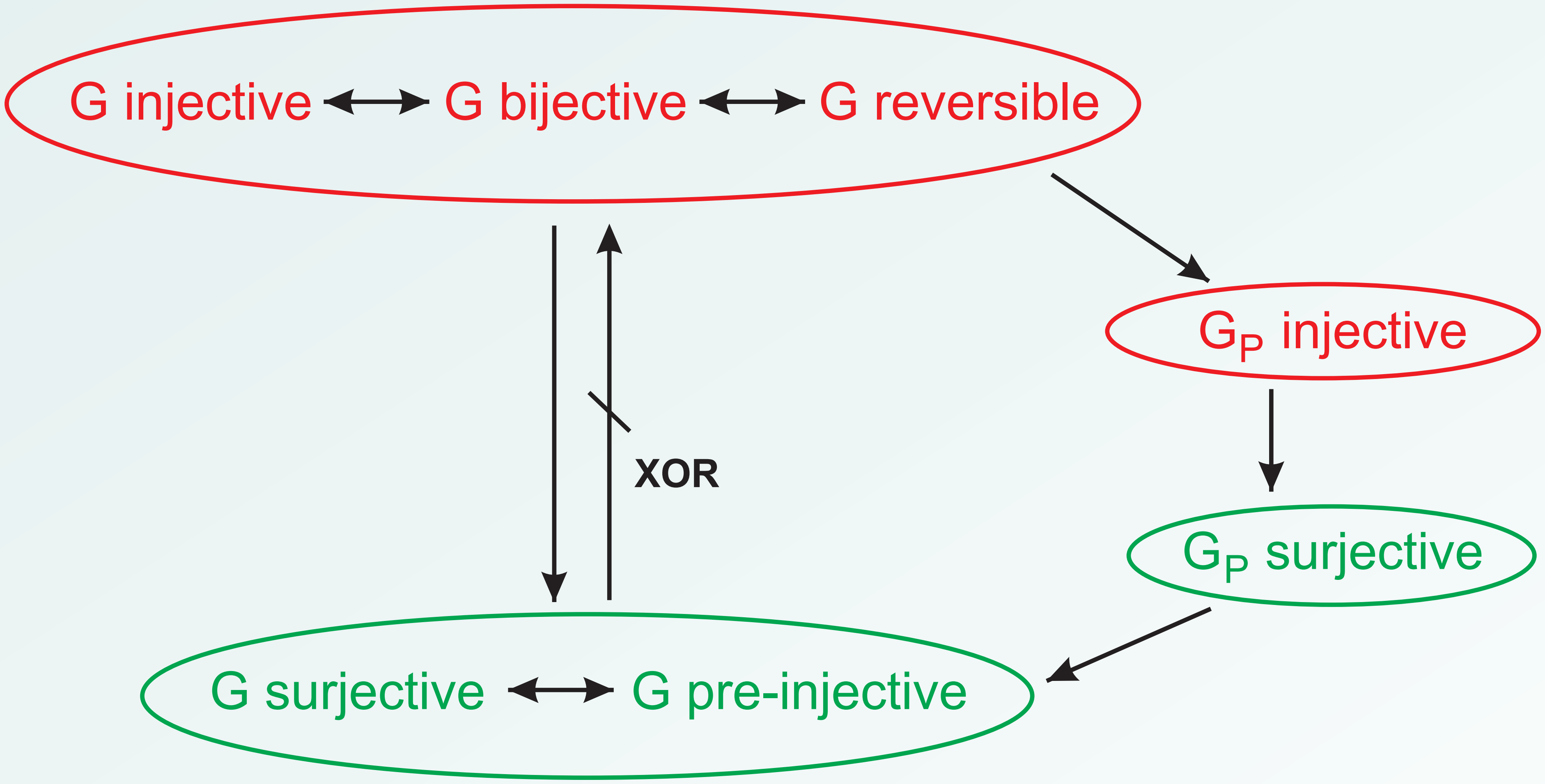
We also have

$$\mathbf{G_P} \text{ injective } \implies \mathbf{G_P} \text{ surjective}$$

Indeed, fix any d linearly independent periods, and let $A \subseteq S^{\mathbb{Z}^d}$ be the set of configurations with these periods. Then

- A is finite,
- G is injective on A,
- $G(A) \subseteq A$.

We conclude that $G(A) = A$, and every periodic configuration has a periodic pre-image.


G injective
G bijective
G reversible
G_P injective
G_P surjective
XOR
G surjective
G pre-injective

Here we get the first **dimension sensitive** property. The following equivalences are only known to hold among one-dimensional CA:

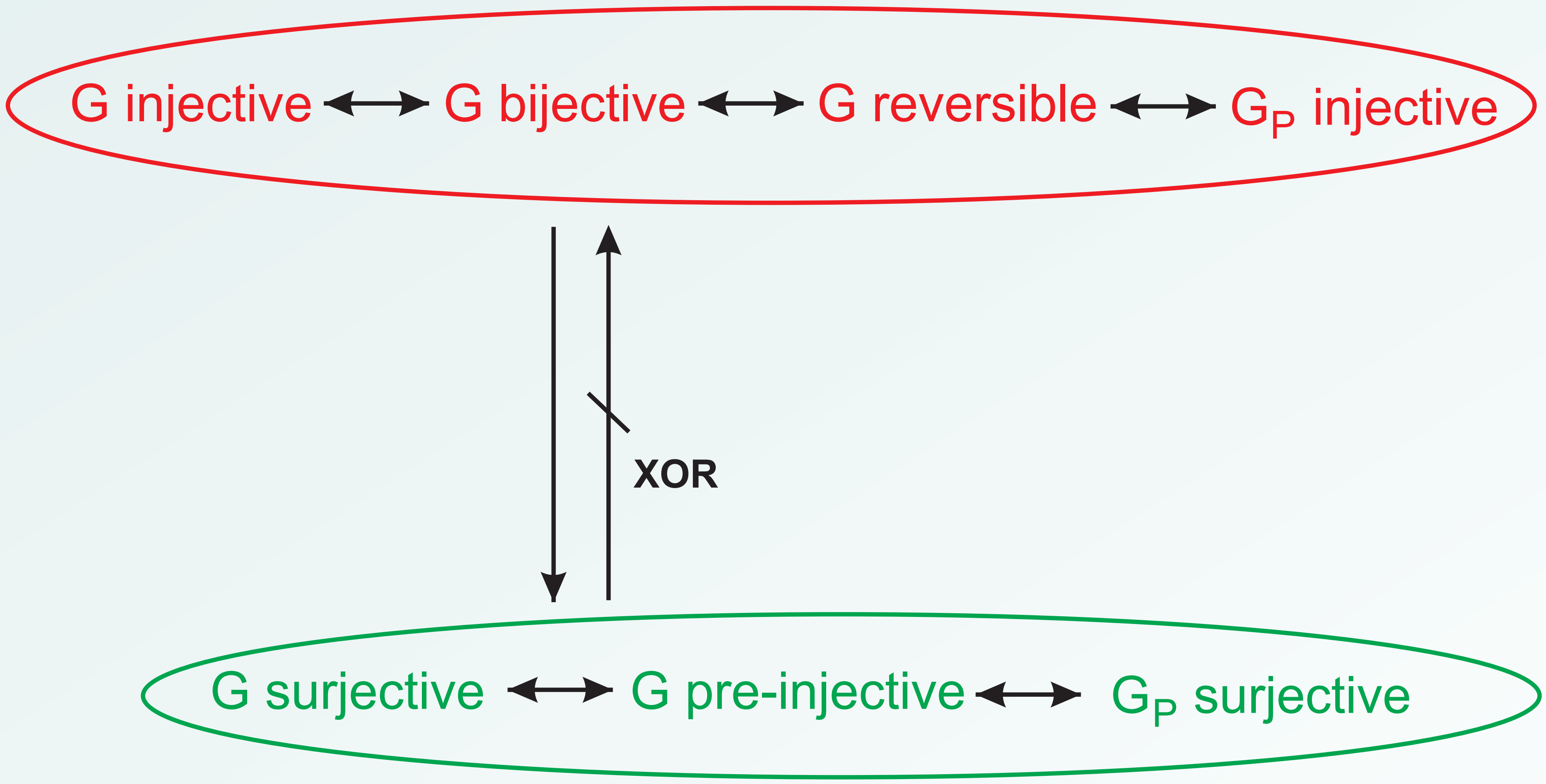
$$\begin{aligned} \mathbf{G} \text{ injective} &\iff \mathbf{G_P} \text{ injective} \\ \mathbf{G} \text{ surjective} &\iff \mathbf{G_P} \text{ surjective} \end{aligned}$$

Here we get the first **dimension sensitive** property. The following equivalences are only known to hold among one-dimensional CA:

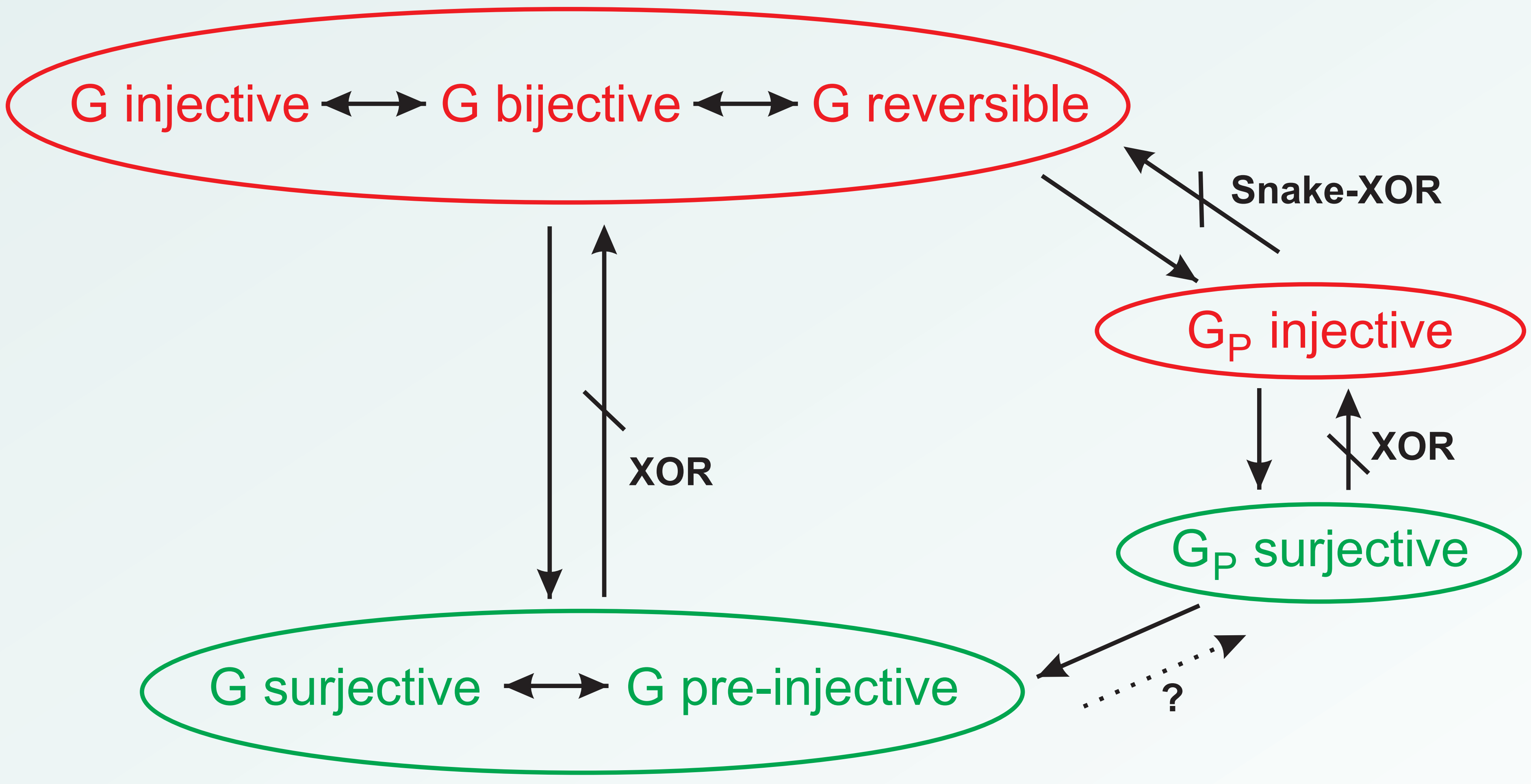
$$\begin{aligned} \mathbf{G} \text{ injective} &\iff \mathbf{G_P} \text{ injective} \\ \mathbf{G} \text{ surjective} &\iff \mathbf{G_P} \text{ surjective} \end{aligned}$$

- The first equivalence is not true among two-dimensional CA: Later we'll see a counter example **Snake-XOR**.
- It is not known whether the second equivalence is true in 2D.

Only in 1D


G injective
G bijective
G reversible
G_P injective
XOR
G surjective
G pre-injective
G_P surjective

In 2D


G injective
G bijective
G reversible
Snake-XOR
G_P injective
XOR
XOR
G_P surjective
G surjective
G pre-injective
?

Note that we have obtained two totally different proofs that injective CA are surjective:

$$\mathbf{G} \text{ injective} \implies \mathbf{G} \text{ pre-injective} \implies \mathbf{G} \text{ surjective}$$

$$\mathbf{G} \text{ injective} \implies \mathbf{G_P} \text{ injective} \implies \mathbf{G_P} \text{ surjective} \implies \mathbf{G} \text{ surjective}$$

Wang tiles and decidability questions

Suppose we are given a cellular automaton and want to know if it is reversible or surjective ? Is there an **algorithm** to determine this ?

Wang tiles and decidability questions

Suppose we are given a cellular automaton and want to know if it is reversible or surjective ? Is there an **algorithm** to determine this ?

It turns out that these algorithmic problems are **undecidable** for two- and higher dimensional CA: **no algorithm exists!**

In contrast, there are algorithms to check reversibility and surjectivity in 1D.

Wang tiles and decidability questions

Suppose we are given a cellular automaton and want to know if it is reversible or surjective ? Is there an **algorithm** to determine this ?

It turns out that these algorithmic problems are **undecidable** for two- and higher dimensional CA: **no algorithm exists!**

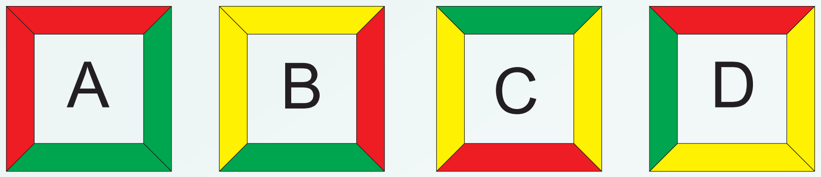
In contrast, there are algorithms to check reversibility and surjectivity in 1D.

A useful tool to obtain undecidability results is the concept of **Wang tiles** and the undecidable **tiling problem**.

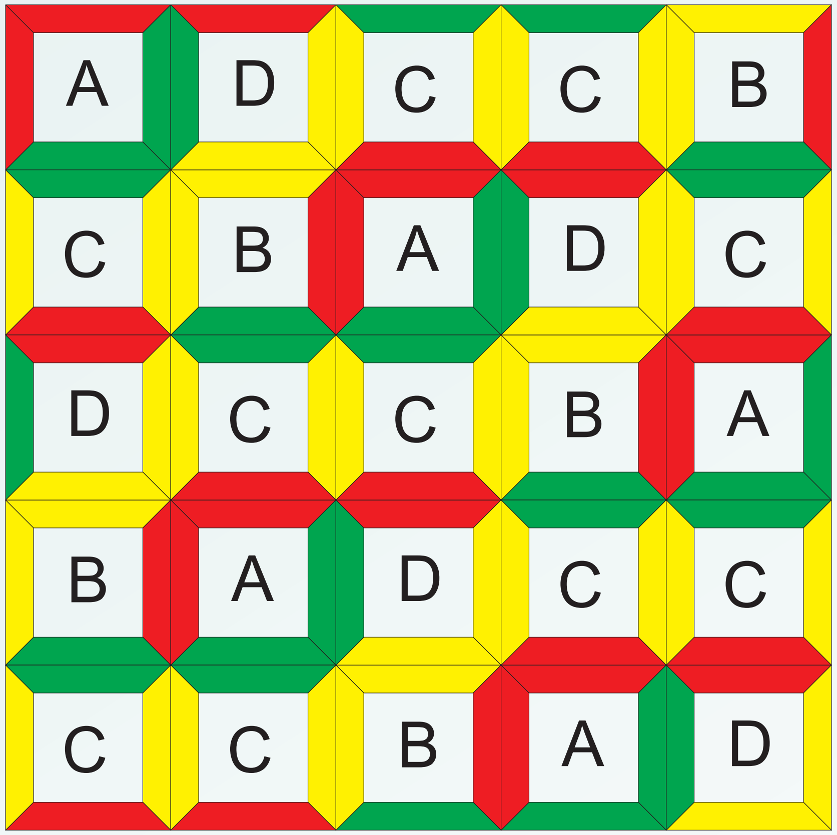
A **Wang tile** is a unit square tile with colored edges. A tile set T is a finite collection of such tiles. A valid tiling is an assignment

$$\mathbb{Z}^2 \longrightarrow T$$

of tiles on infinite square lattice so that the abutting edges of adjacent tiles have the same color.



With copies of the given four tiles we can properly tile a 5×5 square…



… and since the colors on the borders match this square can be repeated to form a valid periodic tiling of the plane.

The **tiling problem** (aka the **Domino problem**) of Wang tiles is the algorithmic problem to determine if a given finite set of Wang tiles admits a valid tiling of the plane.

Facts:

(1) If T admits valid tilings inside squares of arbitrary size then it admits a valid tiling of the whole plane.

Facts:

(1) If T admits valid tilings inside squares of arbitrary size then it admits a valid tiling of the whole plane.

- Follows from compactness of space $T^{\mathbb{Z}^2}$.

Facts:

(2) There exist **aperiodic** tile sets that

- admit valid tilings of the plane, but
- do not admit any periodic tilings.

Facts:

(2) There exist **aperiodic** tile sets that

- admit valid tilings of the plane, but
- do not admit any periodic tilings.

In 1966, R. Berger proved that such aperiodic tile sets exist.
His tile set contains 20426 tiles.

Facts:

(2) There exist **aperiodic** tile sets that

- admit valid tilings of the plane, but
- do not admit any periodic tilings.

In 1966, R. Berger proved that such aperiodic tile sets exist. His tile set contains 20426 tiles.

- R. Robinson (1971) 56 tiles
- R. Amman (1977) 16 tiles
- J. Kari, K. Culik (1996) 14 and 13 tiles
- E. Jeandel, M. Rao (2015) 11 tiles.

Jeandel and Rao show by computer that there is no smaller one.

Berger in fact proved more:

Theorem (R.Berger 1966): The tiling problem of Wang tiles is undecidable.

Berger in fact proved more:

Theorem (R.Berger 1966): The tiling problem of Wang tiles is undecidable.

The tiling problem can be reduced to various decision problems concerning (two-dimensional) cellular automata

$\Longrightarrow$ undecidability of CA problems

This is not so surprising since Wang tilings are **"static"** versions of **"dynamic"** cellular automata.

Berger in fact proved more:

Theorem (R.Berger 1966): The tiling problem of Wang tiles is undecidable.

The tiling problem can be reduced to various decision problems concerning (two-dimensional) cellular automata

$\Longrightarrow$ undecidability of CA problems

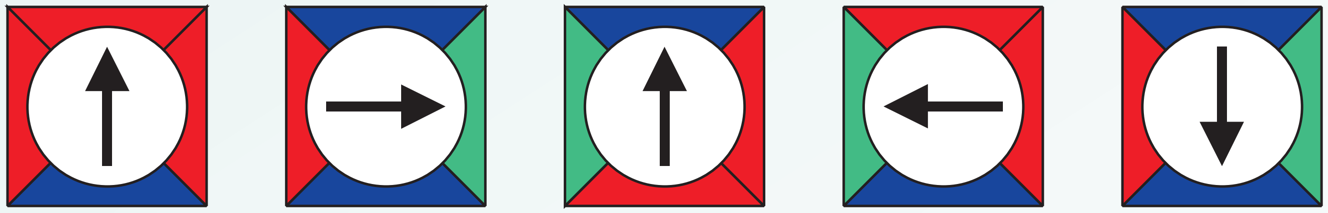
This is not so surprising since Wang tilings are **”static”** versions of **”dynamic”** cellular automata.

In the following: **tiling problem $\Longrightarrow$ 2D CA reversibility**

SNAKES

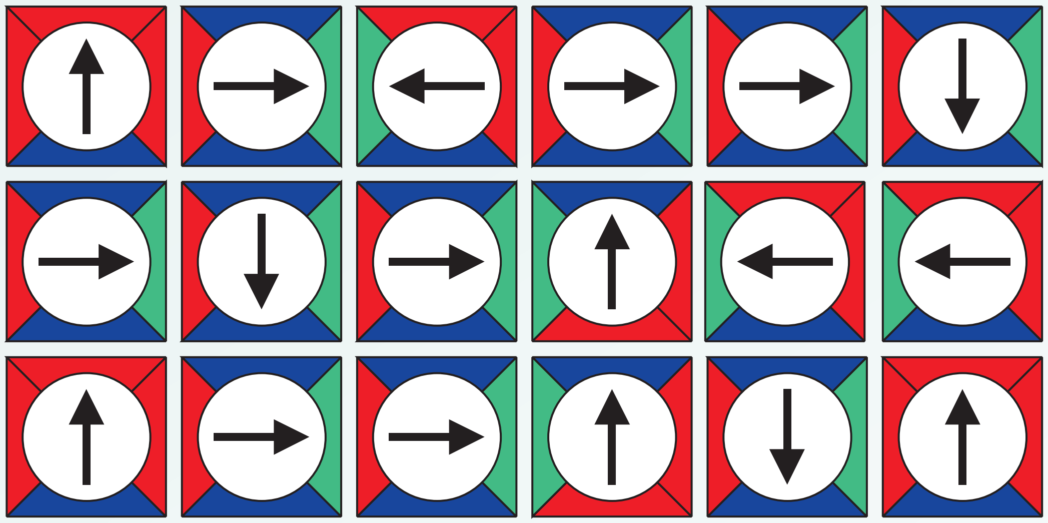
SNAKES is a tile set with some interesting (and useful) properties.

In addition to colored edges, these tiles also have an arrow printed on them. The arrow is horizontal or vertical and it points to one of the four neighbors of the tile:

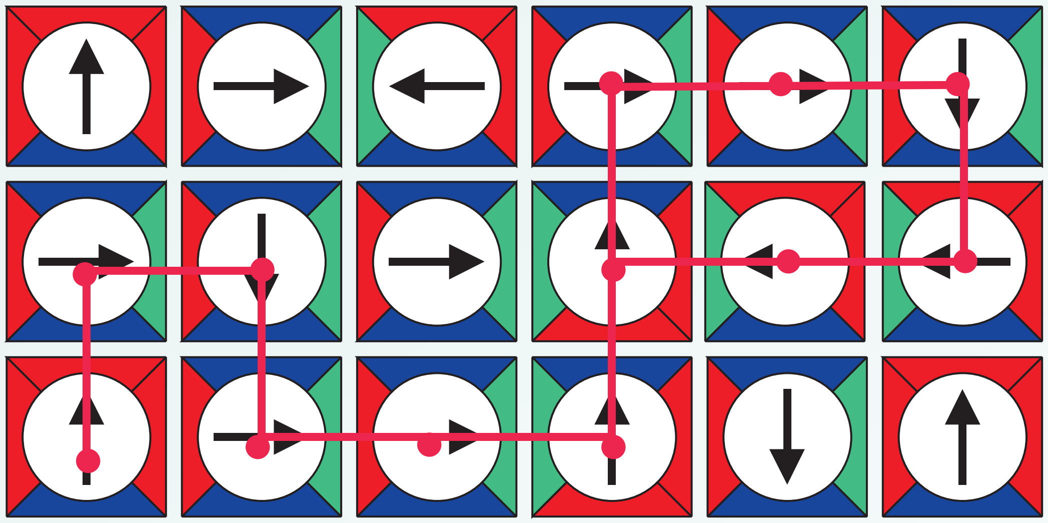


Such tiles with arrows are called **directed tiles**.

Given a configuration (valid tiling or not!) and a starting position, the arrows specify a path on the plane. Each position is followed by the neighboring position indicated by the arrow of the tile:

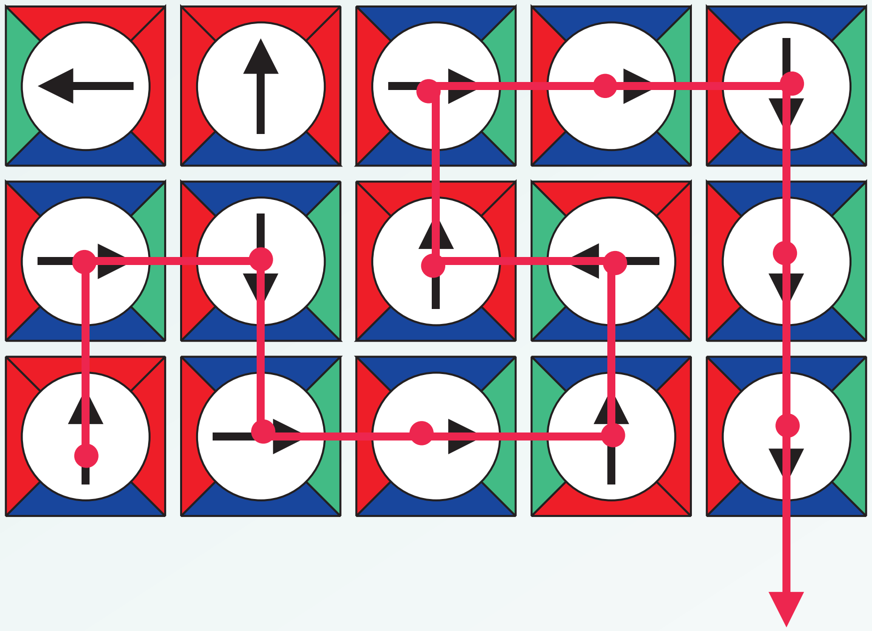


Given a configuration (valid tiling or not!) and a starting position, the arrows specify a path on the plane. Each position is followed by the neighboring position indicated by the arrow of the tile:



The path may enter a loop…

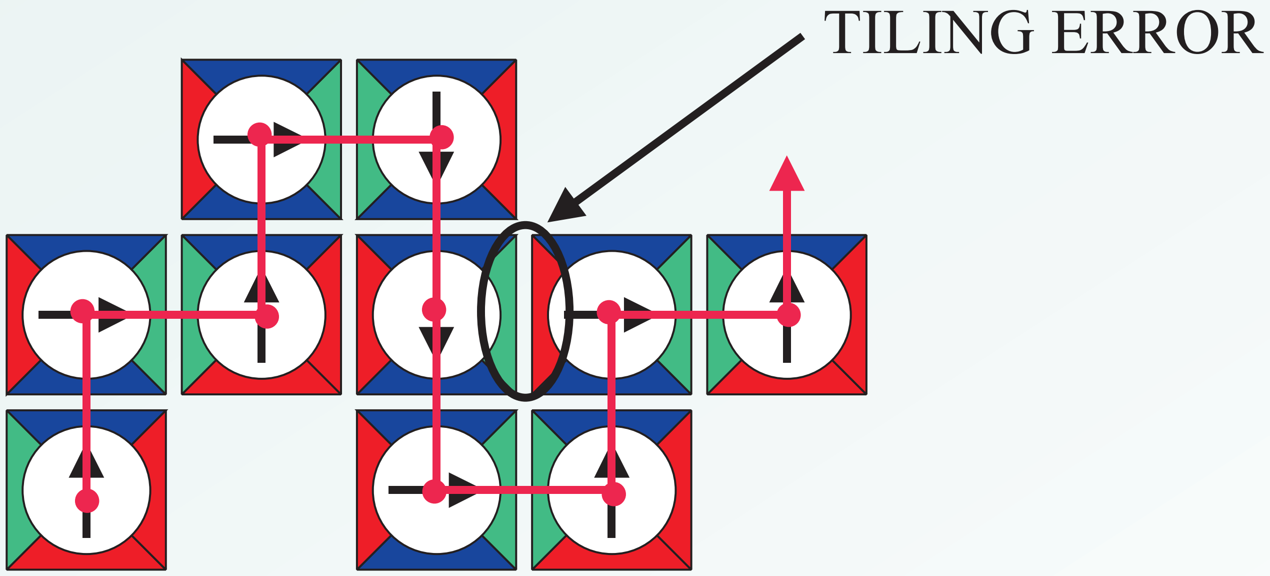
Given a configuration (valid tiling or not!) and a starting position, the arrows specify a path on the plane. Each position is followed by the neighboring position indicated by the arrow of the tile:



…or the path may be infinite and never return to a tile visited before.

The directed tile set SNAKES has the following property: On any configuration (valid tiling or not) and on any path that follows the arrows one of the following two things happens:

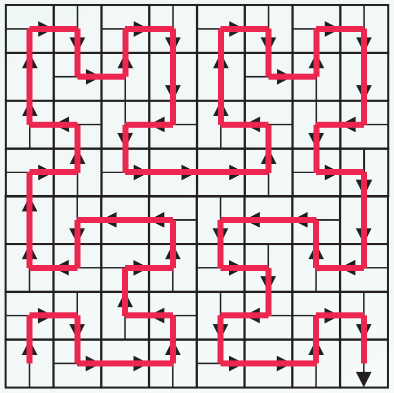
(1) Either there is a tiling error between two tiles both of which are on the path,



The directed tile set SNAKES has the following property: On any configuration (valid tiling or not) and on any path that follows the arrows one of the following two things happens:

(1) Either there is a tiling error between two tiles both of which are on the path,

(2) or the path is a plane-filling path, that is, for every positive integer n there exists an $n \times n$ square all of whose positions are visited by the path.



The directed tile set SNAKES has the following property: On any configuration (valid tiling or not) and on any path that follows the arrows one of the following two things happens:

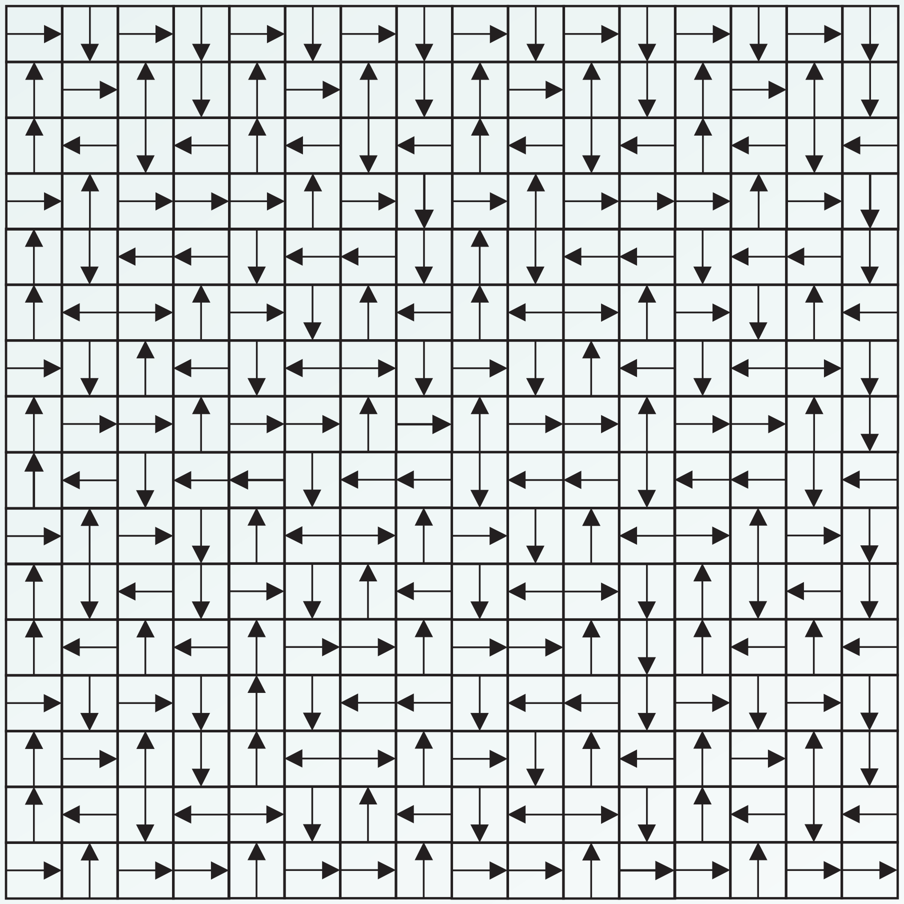
(1) Either there is a tiling error between two tiles both of which are on the path,

(2) or the path is a plane-filling path, that is, for every positive integer n there exists an $n \times n$ square all of whose positions are visited by the path.

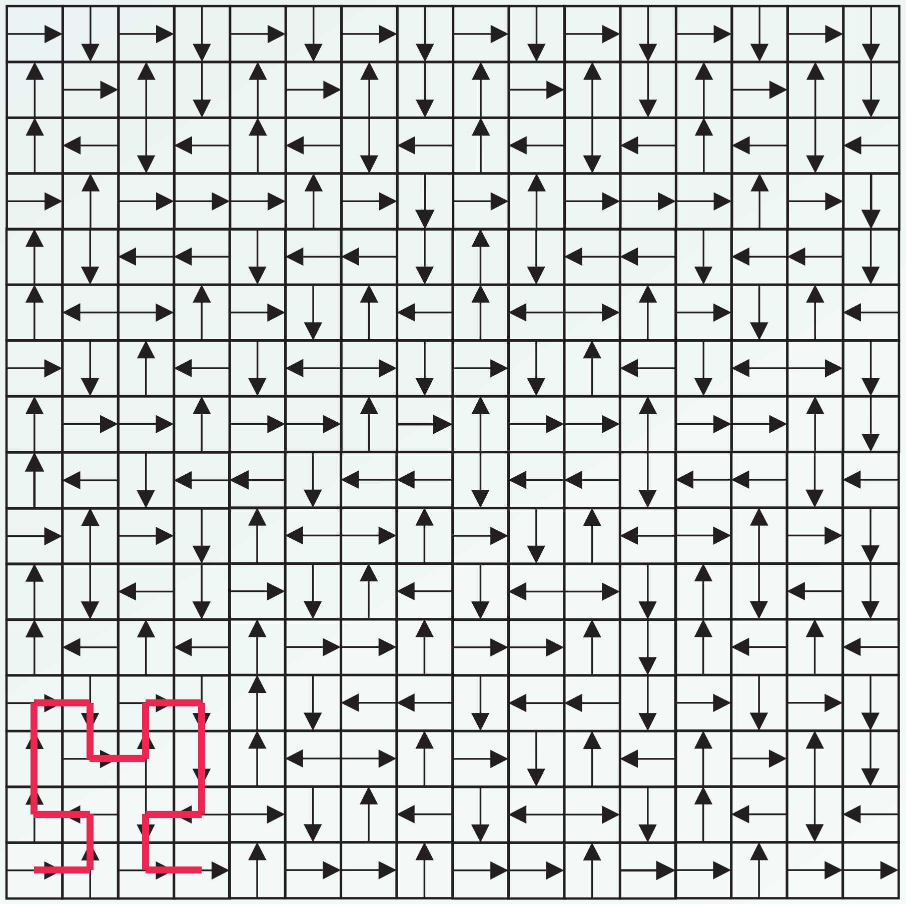
Note that the tiling may be invalid outside path P, yet the path is forced to snake through larger and larger squares.

SNAKES also has the property that it admits a valid tiling.

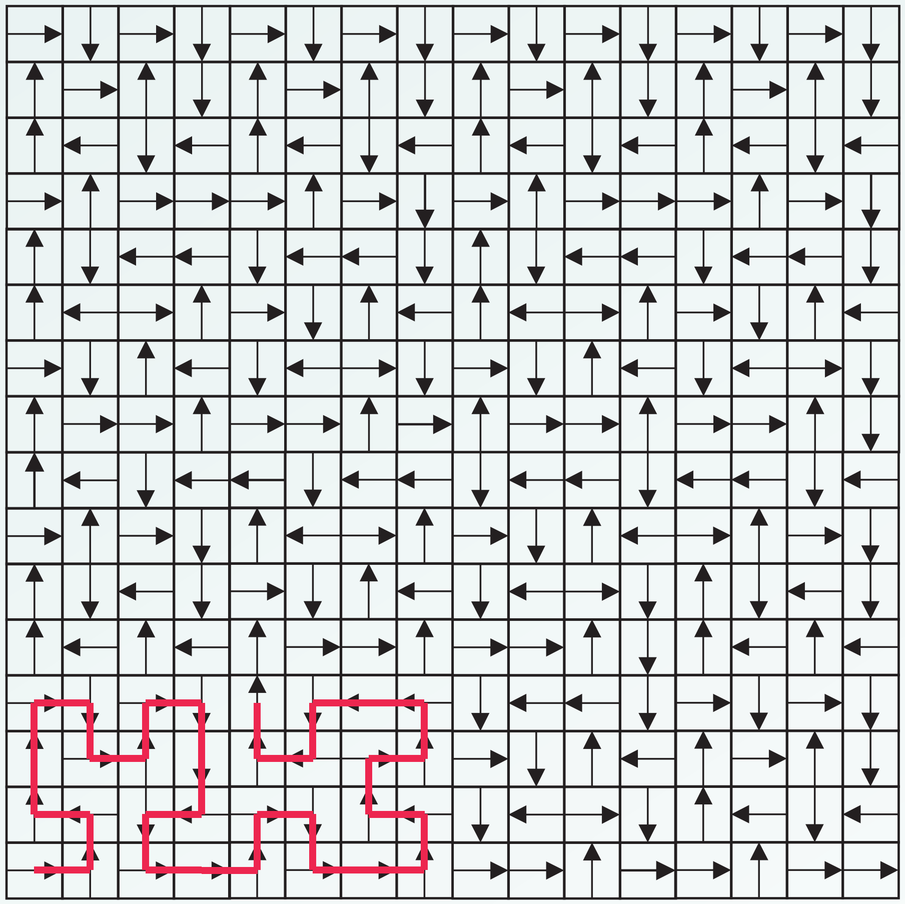
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



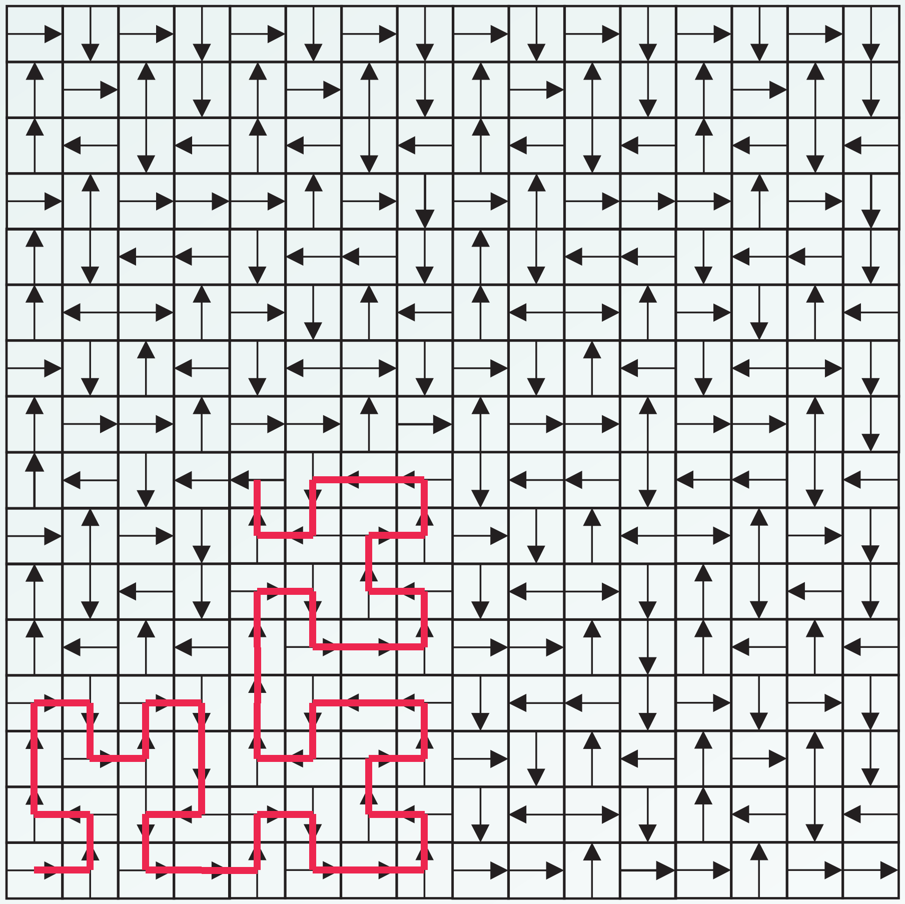
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



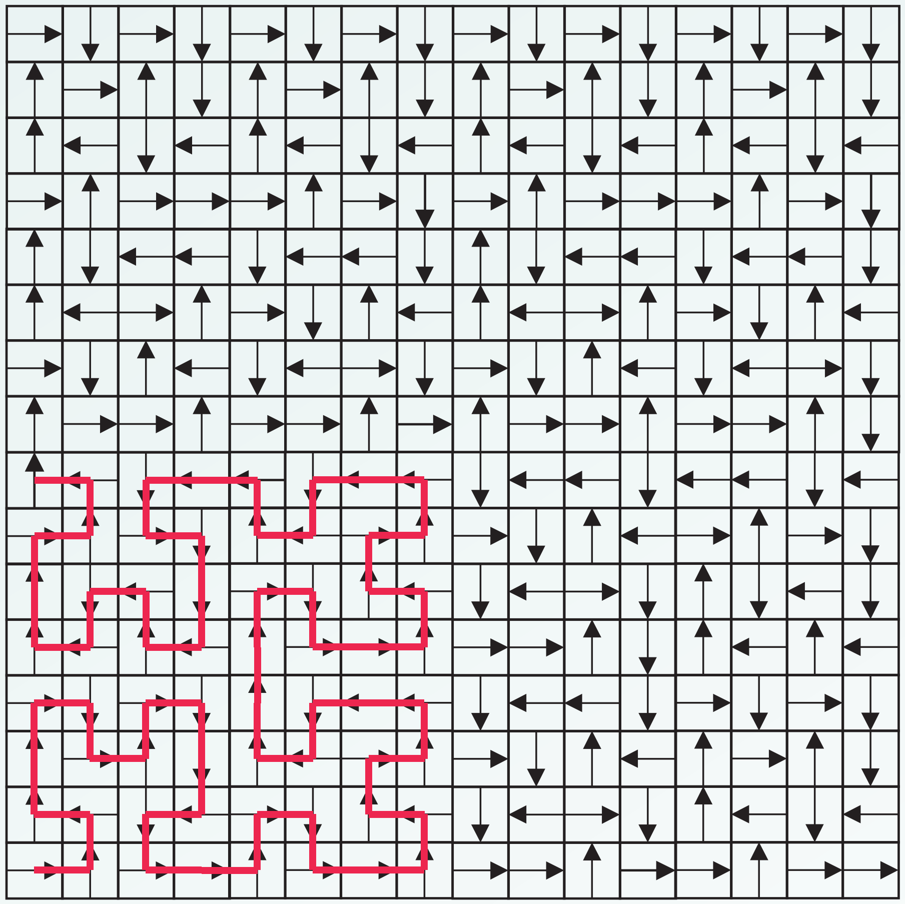
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



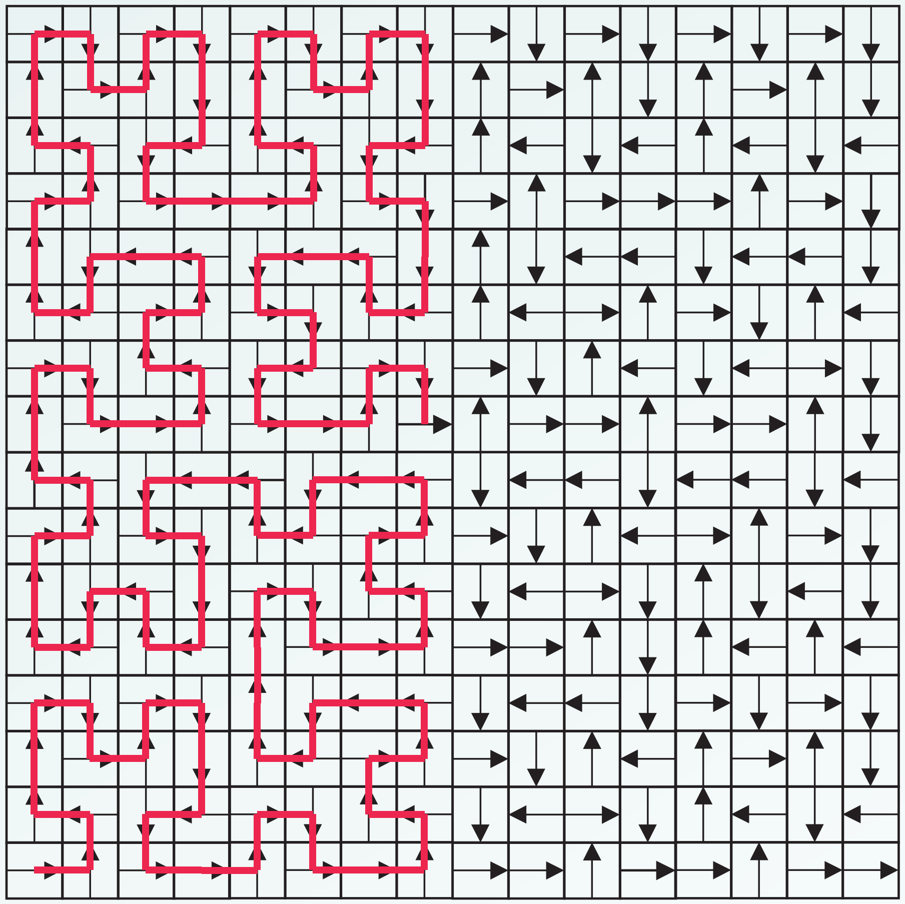
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



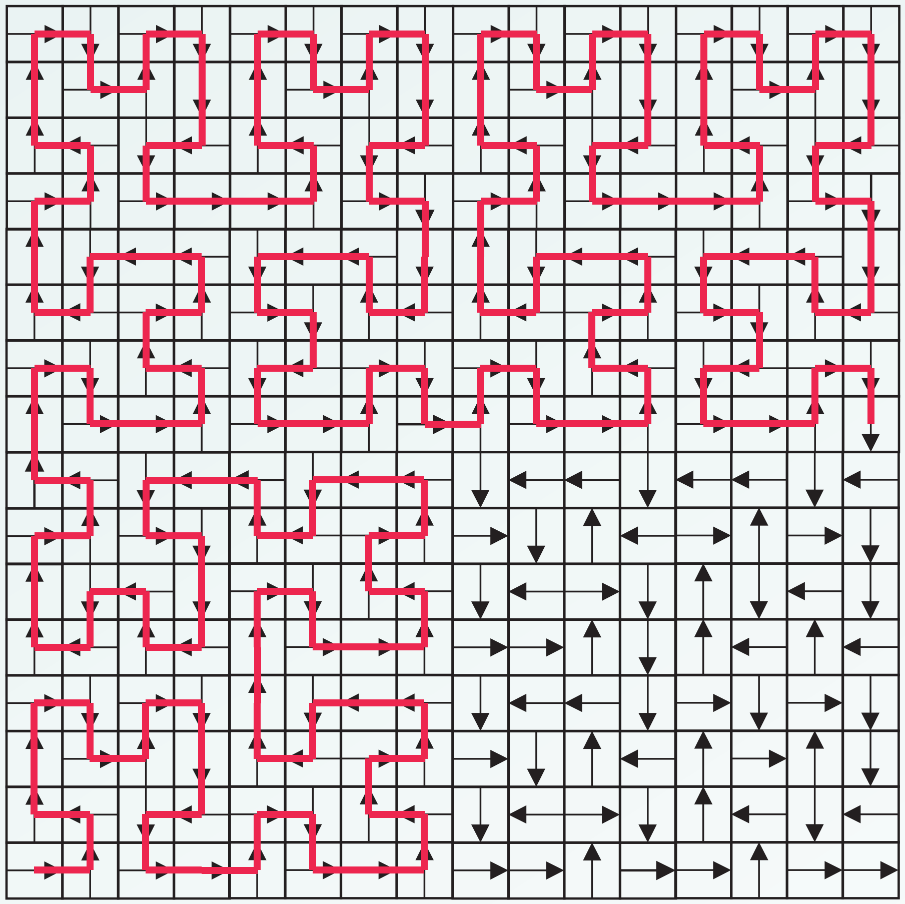
The paths that Snakes forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



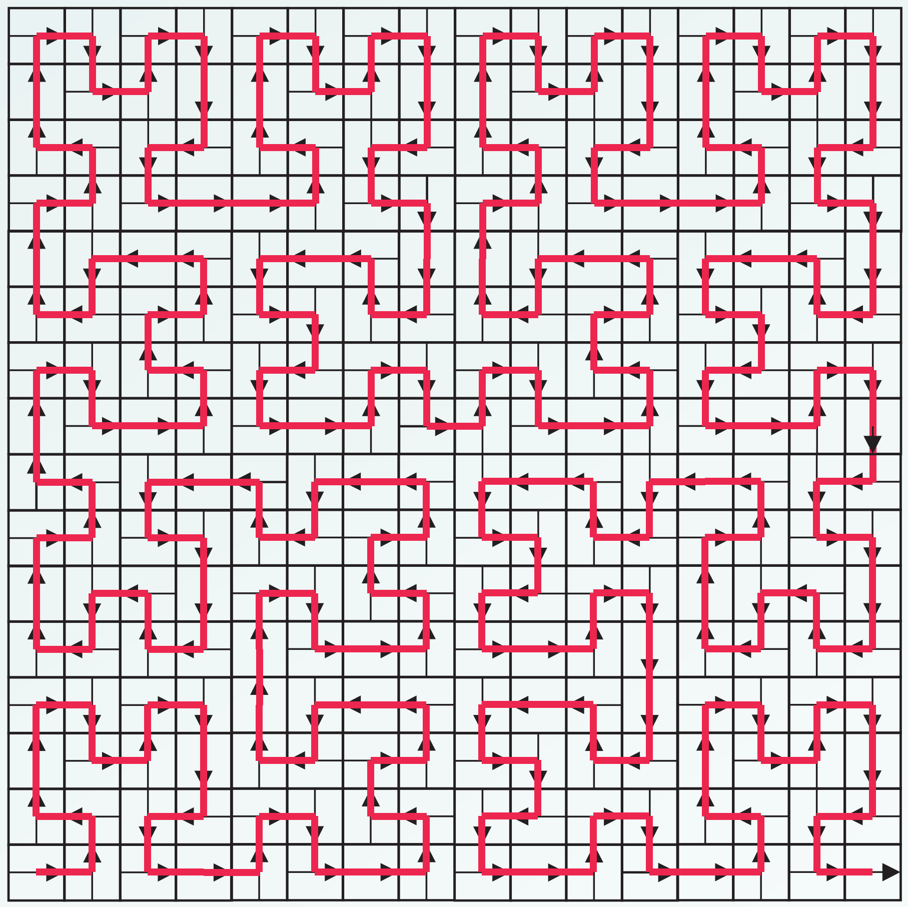
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



Applications of Snakes

First application of Snakes: An example of a two-dimensional CA that is injective on periodic configurations but is not injective on all configurations.

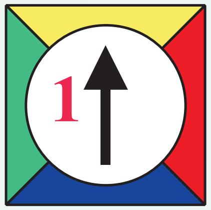
The **Snake XOR** CA confirms that in 2D

$$G \text{ injective} \quad \not\Leftarrow \quad G_P \text{ injective.}$$

The state set of the CA is

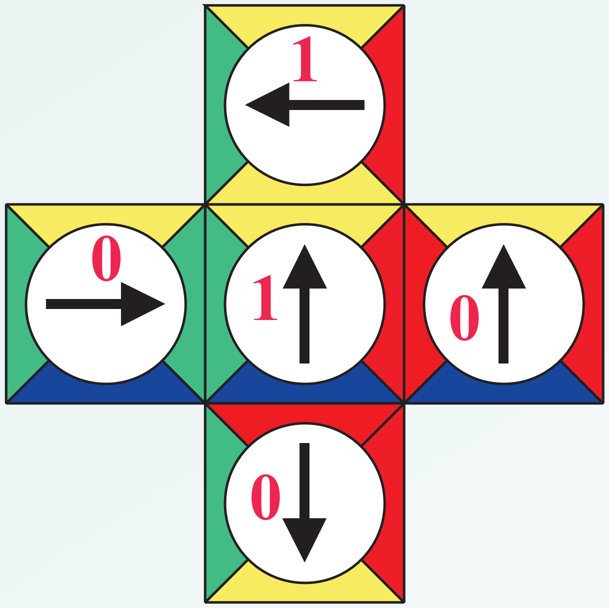
$$S = \textsc{Snakes} \times \{0, 1\}.$$

(Each snake tile is attached a red bit.)



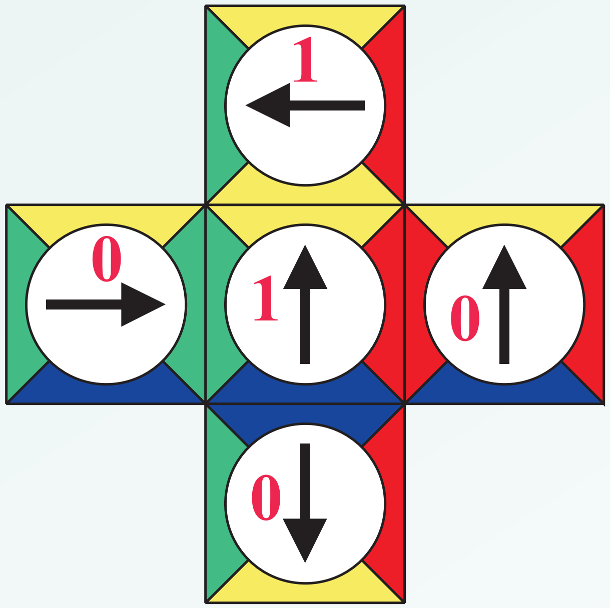
The local rule checks whether the tiling is valid at the cell:

- If there is a tiling error, no change in the state.



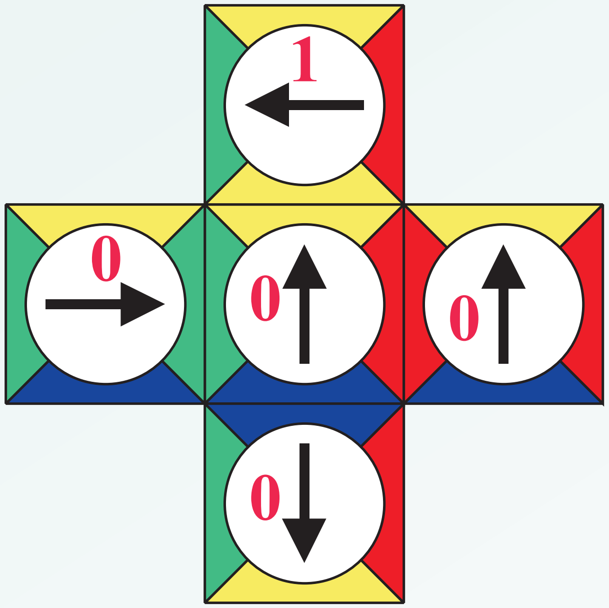
The local rule checks whether the tiling is valid at the cell:

- If there is a tiling error, no change in the state.
- If the tiling is valid, the cell is **active**: the bit of the neighbor next on the path is XOR'ed to the bit of the cell.



The local rule checks whether the tiling is valid at the cell:

- If there is a tiling error, no change in the state.
- If the tiling is valid, the cell is **active**: the bit of the neighbor next on the path is XOR'ed to the bit of the cell.



Snake XOR is not injective:

The following two configurations have the same successor: The SNAKES tilings of the configurations form the same valid tiling of the plane. In one of the configurations all bits are set to 0, and in the other configuration all bits are 1.

All cells are active because the tilings are correct. This means that all bits in both configurations become 0. So the two configurations become identical. The CA is not injective.

Snake XOR is injective on periodic configurations:

Suppose there are different periodic configurations c and d with the same successor. Since only bits may change, c and d must have identical SNAKES tiles everywhere. So they must have different bits 0 and 1 in some position $\vec{p}_1 \in \mathbb{Z}^2$.

Snake XOR is injective on periodic configurations:

Suppose there are different periodic configurations c and d with the same successor. Since only bits may change, c and d must have identical Snakes tiles everywhere. So they must have different bits 0 and 1 in some position $\vec{p}_1 \in \mathbb{Z}^2$.

Because c and d have identical successors:

- The cell in position $\vec{p}_1$ must be active, that is, the Snakes tiling is valid in position $\vec{p}_1$.
- The bits stored in the next position $\vec{p}_2$ (indicated by the direction) are different in c and d.

Snake XOR is injective on periodic configurations:

Suppose there are different periodic configurations c and d with the same successor. Since only bits may change, c and d must have identical SNAKES tiles everywhere. So they must have different bits 0 and 1 in some position $\vec{p}_1 \in \mathbb{Z}^2$.

Because c and d have identical successors:

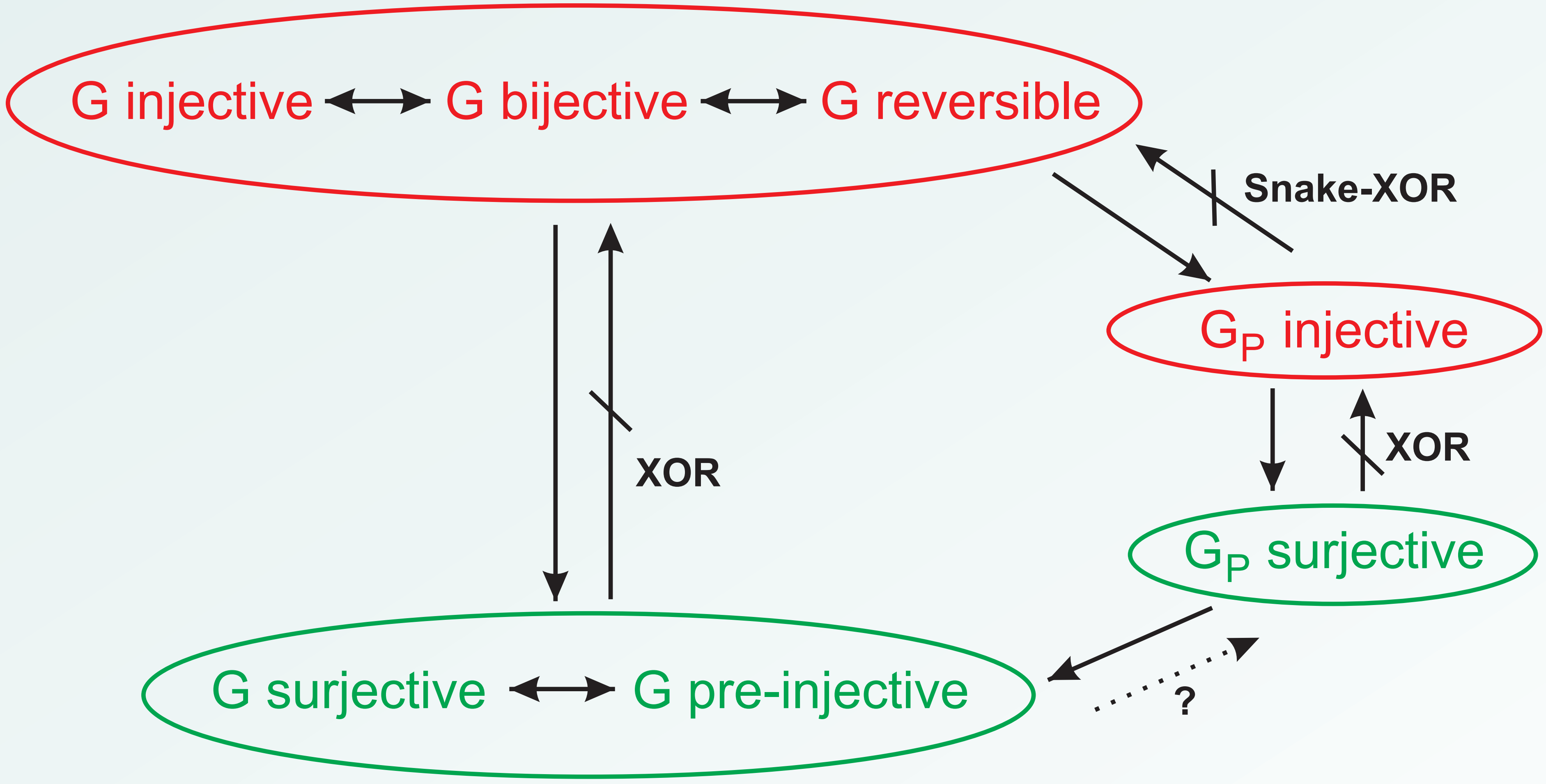
- The cell in position $\vec{p}_1$ must be active, that is, the SNAKES tiling is valid in position $\vec{p}_1$.
- The bits stored in the next position $\vec{p}_2$ (indicated by the direction) are different in c and d.

Hence we can repeat the reasoning in position $\vec{p}_2$.

The same reasoning can be repeated over and over again. The positions $\vec{p}_1, \vec{p}_2, \vec{p}_3, \ldots$ form a path that follows the arrows on the tiles. There is no tiling error at any tile on this path.

But this contradicts the fact that the plane filling property of SNAKES guarantees that on periodic configuration every path encounters a tiling error. $\square$

In 2D



G injective
G bijective
G reversible
Snake-XOR
G$_P$ injective
XOR
XOR
G$_P$ surjective
?
G surjective
G pre-injective

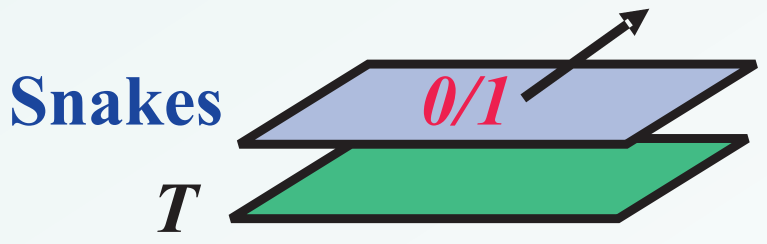
Second application of SNAKES: It is undecidable to determine if a given two-dimensional CA is reversible.

Second application of SNAKES: It is undecidable to determine if a given two-dimensional CA is reversible.

The proof is a reduction from the tiling problem, using the tile set SNAKES.

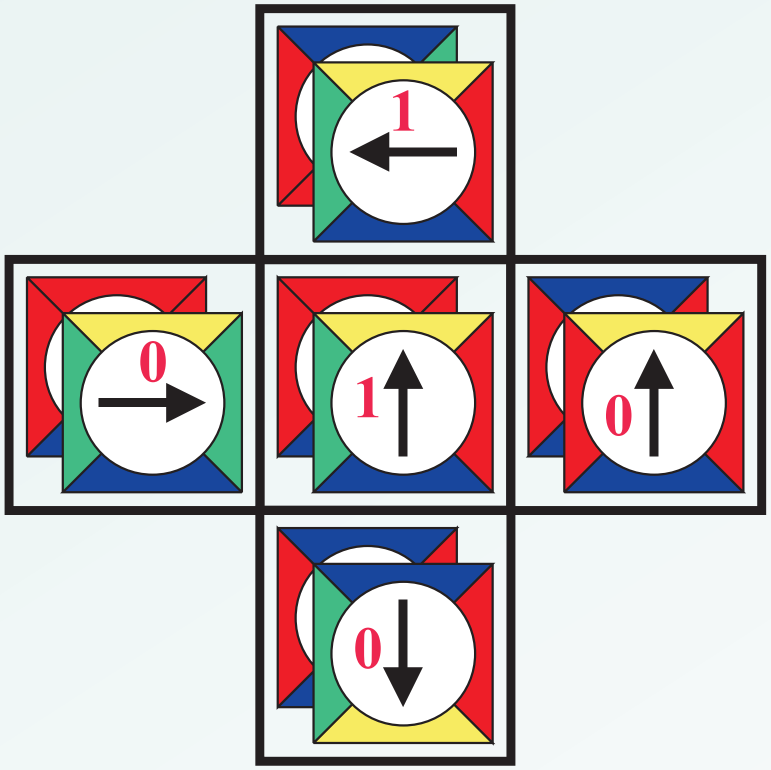
For any given tile set T we construct a CA with the state set

$$S = T \times \textsc{Snakes} \times \{0, 1\}.$$



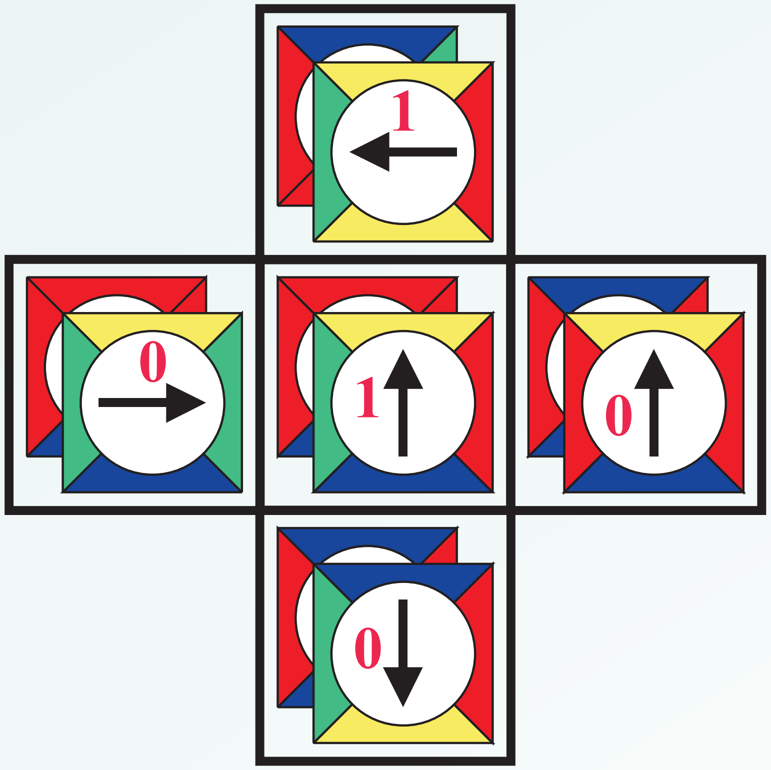
The local rule is analogous to **Snake XOR** with the difference that the correctness of the tiling is checked in both tile layers:

- If there is a tiling error then the cell is inactive.



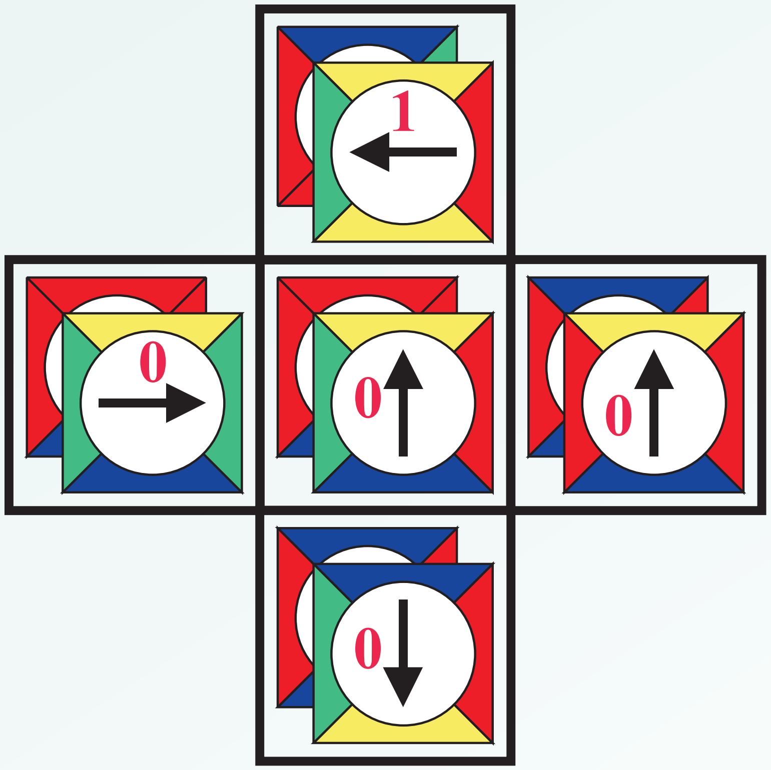
The local rule is analogous to **Snake XOR** with the difference that the correctness of the tiling is checked in both tile layers:

- If there is a tiling error then the cell is inactive.
- If both tilings are valid, the bit of the neighbor next on the path is XOR'ed to the bit of the cell.



The local rule is analogous to **Snake XOR** with the difference that the correctness of the tiling is checked in both tile layers:

- If there is a tiling error then the cell is inactive.
- If both tilings are valid, the bit of the neighbor next on the path is XOR'ed to the bit of the cell.



We can reason exactly as with **Snake XOR**, and show that the CA is reversible if and only if the tile set T does not admit a plane tiling:

We can reason exactly as with **Snake XOR**, and show that the CA is reversible if and only if the tile set T does not admit a plane tiling:

$(\Longrightarrow)$ If a valid tiling of the plane exists then we can construct two different configurations of the CA that have the same image under G. The SNAKES and the T layers of the configurations form the same valid tilings of the plane. In one of the configurations all bits are 0, and in the other configuration all bits are 1.

All cells are active because the tilings are correct. This means that all bits in both configurations become 0. So the two configurations become identical. The CA is not injective.

($\Longleftarrow$) Conversely, assume that the CA is not injective. Let c and d be two different configurations with the same successor. Since only bits may change, c and d must have identical SNAKES and T layers. So they must have different bits 0 and 1 in some position $\vec{p}_1 \in \mathbb{Z}^2$.

($\Longleftarrow$) Conversely, assume that the CA is not injective. Let c and d be two different configurations with the same successor. Since only bits may change, c and d must have identical SNAKES and T layers. So they must have different bits 0 and 1 in some position $\vec{p}_1 \in \mathbb{Z}^2$.

Because c and d have identical successors:

- The cell in position $\vec{p}_1$ must be active, that is, the SNAKES and T tilings are both valid in position $\vec{p}_1$.
- The bits stored in the next position $\vec{p}_2$ (indicated by the direction) are different in c and d.

($\Longleftarrow$) Conversely, assume that the CA is not injective. Let c and d be two different configurations with the same successor. Since only bits may change, c and d must have identical SNAKES and T layers. So they must have different bits 0 and 1 in some position $\vec{p}_1 \in \mathbb{Z}^2$.

Because c and d have identical successors:

- The cell in position $\vec{p}_1$ must be active, that is, the SNAKES and T tilings are both valid in position $\vec{p}_1$.
- The bits stored in the next position $\vec{p}_2$ (indicated by the direction) are different in c and d.

Hence we can repeat the reasoning in position $\vec{p}_2$.

The same reasoning can be repeated over and over again. The positions $\vec{p}_1, \vec{p}_2, \vec{p}_3, \ldots$ form a path that follows the arrows on the tiles. There is no tiling error at any tile on this path so the special property of SNAKES forces the path to cover arbitrarily large squares.

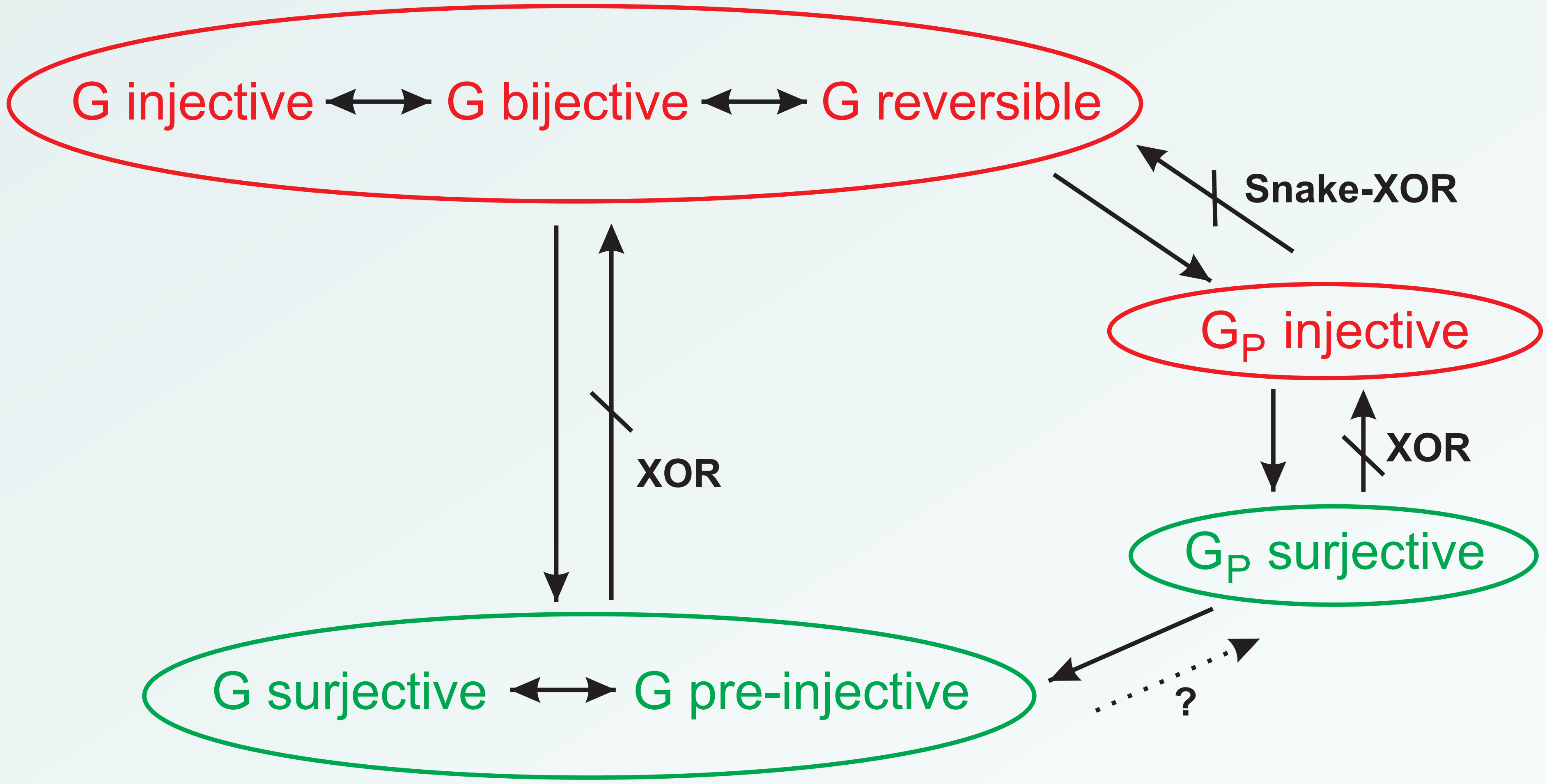
Hence T admits tilings of arbitrarily large squares, and consequently a tiling of the infinite plane. $\square$

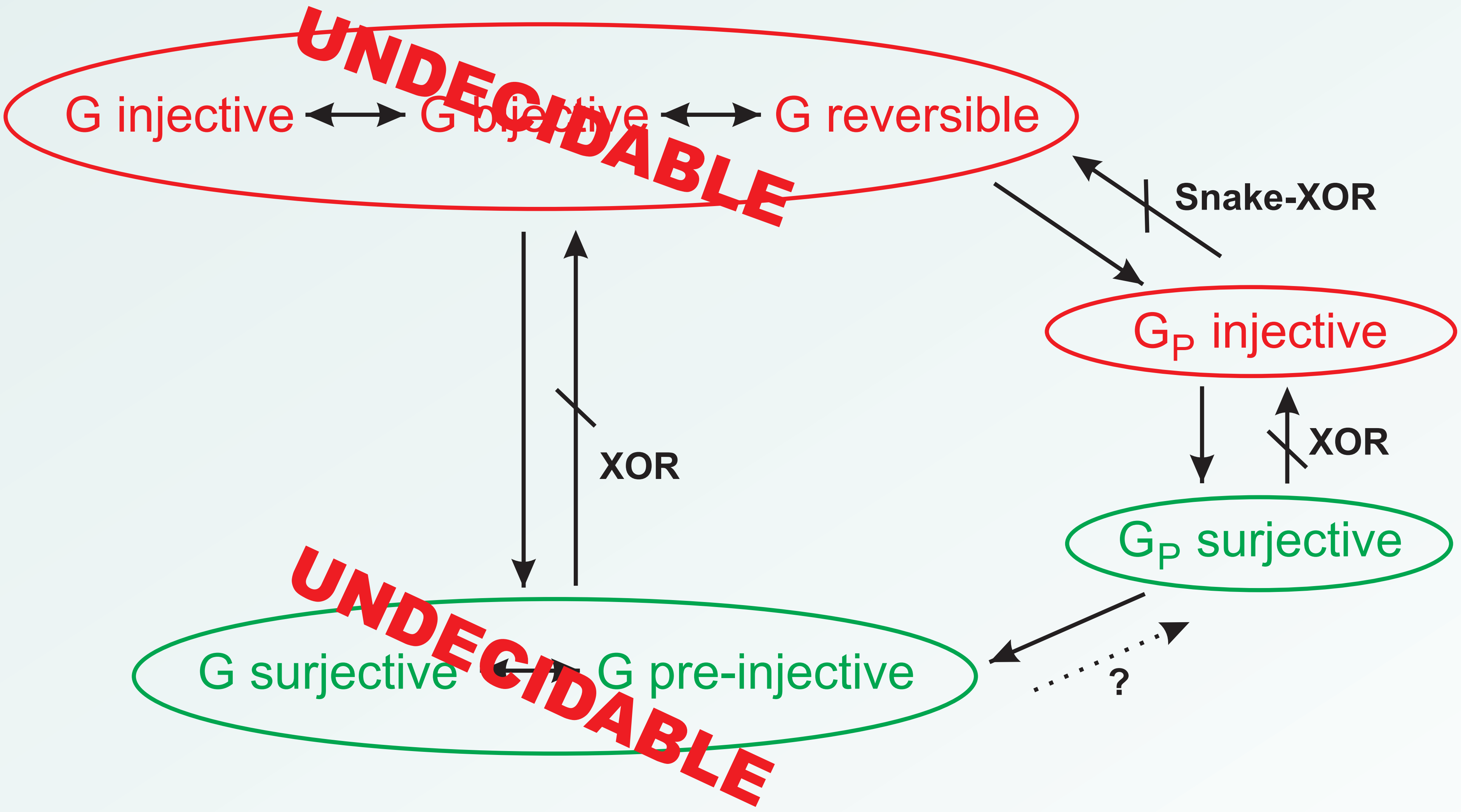
Theorem: It is undecidable whether a given two-dimensional CA is reversible.

Theorem: It is undecidable whether a given two-dimensional CA is reversible.

An analogous (but simpler!) construction can be made for the surjectivity problem, based on the fact surjectivity is equivalent to pre-injectivity:

Theorem: It is undecidable whether a given two-dimensional CA is surjective.


G injective
G bijective
G reversible
Snake-XOR
G_P injective
XOR
XOR
G_P surjective
G surjective
G pre-injective
?


G injective
G bijective
G reversible
UNDECIDABLE
Snake-XOR
G_P injective
XOR
XOR
G_P surjective
G surjective
G pre-injective
UNDECIDABLE
?

Undecidability of reversibility implies the following:

There are some reversible CA that use von Neumann neighborhood but whose inverse automata use a very large neighborhood: There can be no computable upper bound on the extend of this inverse neighborhood.

Undecidability of reversibility implies the following:

There are some reversible CA that use von Neumann neighborhood but whose inverse automata use a very large neighborhood: There can be no computable upper bound on the extend of this inverse neighborhood.

Topological arguments $\Longrightarrow$ A finite neighborhood is enough to determine the previous state of a cell.

Computation theory $\Longrightarrow$ This neighborhood may be extremely large.

Undecidability of surjectivity implies the following:

There are non-surjective CA whose smallest orphan is very large: There can be no computable upper bound on the extend of the smallest orphan.

Undecidability of surjectivity implies the following:

There are non-surjective CA whose smallest orphan is very large: There can be no computable upper bound on the extend of the smallest orphan.

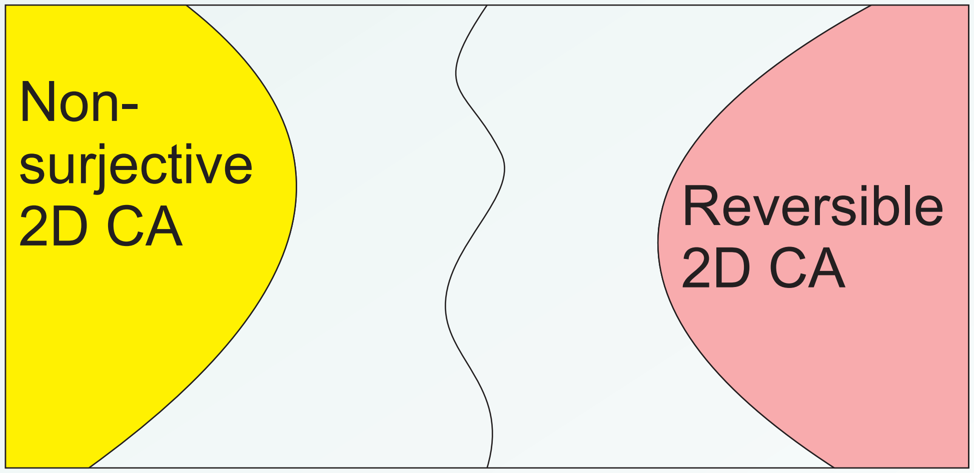
So while the smallest known orphan for Game-Of-Life is pretty big (88 cells), this pales in comparison with some other CA.

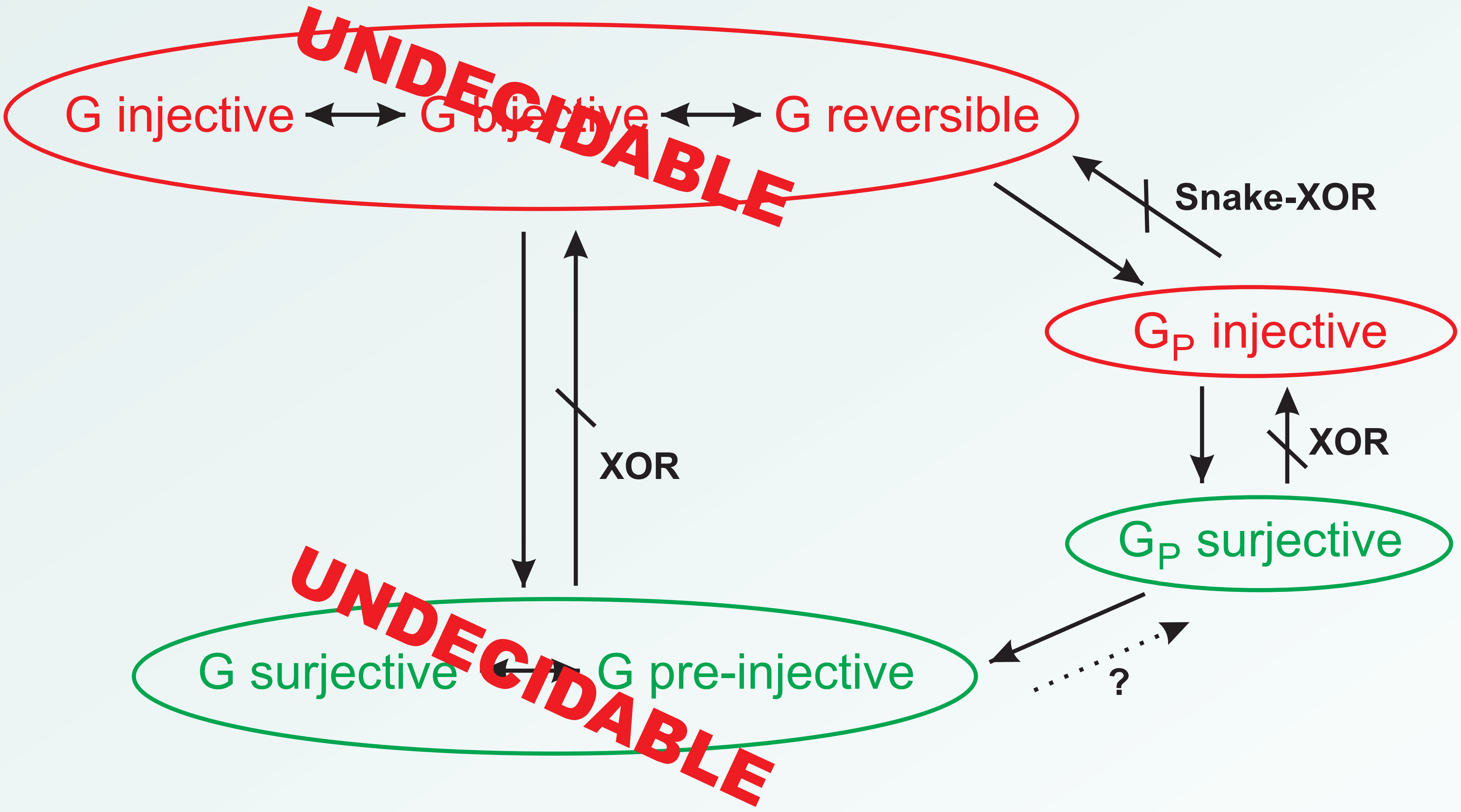
The undecidability proofs for reversibility and surjectivity can be merged into

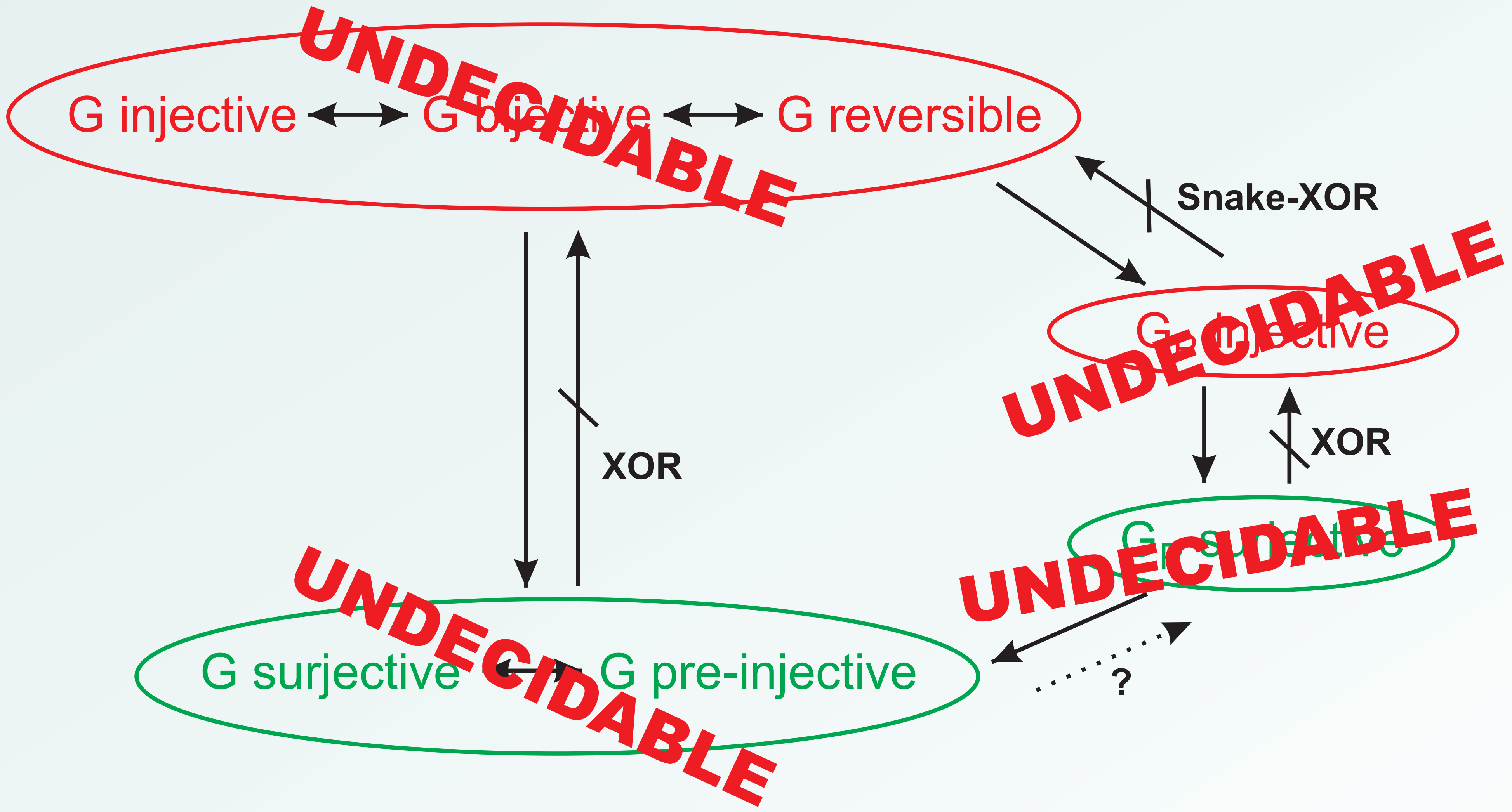
Theorem: The classes of

- Reversible 2D CA
- Non-surjective 2D CA

are recursively inseparable




G injective
G bijective
G reversible
UNDECIDABLE
Snake-XOR
G_P injective
XOR
XOR
G_P surjective
?
G surjective
G pre-injective
UNDECIDABLE


UNDECIDABLE
G injective
G bijective
G reversible
Snake-XOR
UNDECIDABLE
G_P injective
XOR
XOR
UNDECIDABLE
G_P surjective
?
UNDECIDABLE
G surjective
G pre-injective

What about in 1D ?

What about in 1D ?

Both reversibility and surjectivity can be easily decided among one-dimensional CA:

Theorem (Amoroso, Patt 1972): It is decidable whether a given one-dimensional CA is reversible or surjective.

What about in 1D ?

Both reversibility and surjectivity can be easily decided among one-dimensional CA:

Theorem (Amoroso, Patt 1972): It is decidable whether a given one-dimensional CA is reversible or surjective.

We also know the tight bound on the extend of the **inverse neighborhood** in 1D: If a reversible CA has n states and uses two closest neighbors then the neighborhood of the inverse automaton consists of at most $n - 1$ consecutive cells.

Conclusion

Reversible CA are relevant in

- modeling systems in microscopic physics (physics is reversible),
- energy efficient massively parallel computation (universality of reversible CA),
- is symbolic dynamics as automorphisms of the full shift

We learned classical properties

- Hedlund's theorem
- Balance (=invariance of uniform probability distribution)
- Garden-Of-Eden theorem

as well as universality and decibability aspects.