STACS 2016, Orléans

Tutorial on Cellular Automata and Tilings

Jarkko Kari
Department of Mathematics and Statistics
University of Turku

Lecture 1: Tutorial on Cellular automata

- Introduction and examples
- General definitions
- Topolgy & Curtis-Hedlund-Lyndon -theorem
- Reversible CA
- Surjective CA: balance, Garden-of-Eden -theorem

Cellular Automata (CA): Introduction

Cellular automata are among the oldest models of natural computing, studied

- **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**).

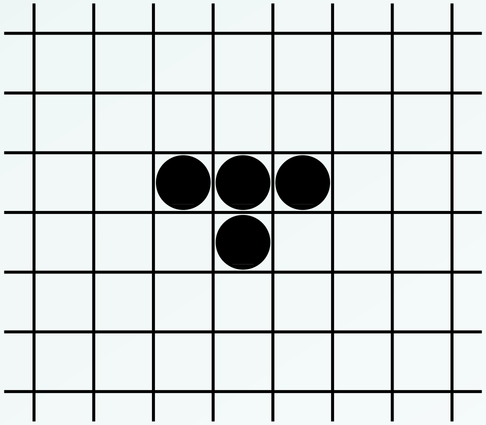
- At each discrete time step each cell counts the number of living cells surrounding it, and based on this number determines its new state.

- All cells change their state simultaneously.

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

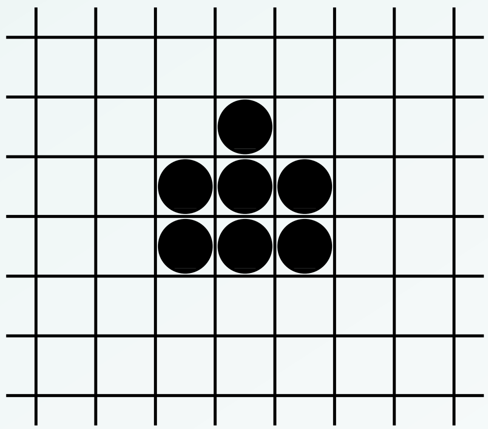
- If the cell is **alive** then it stays alive (survives) iff it has two or three live neighbors. Otherwise it dies of loneliness or overcrowding.

- If the cell is **dead** then it becomes alive iff it has exactly three living neighbors.



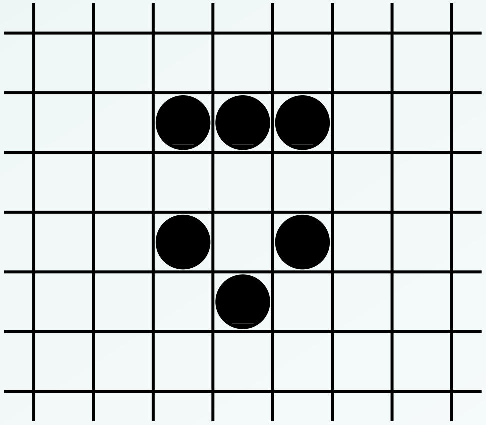
The local update rule asks each cell to check the present states of the eight surrounding cells.

- 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.



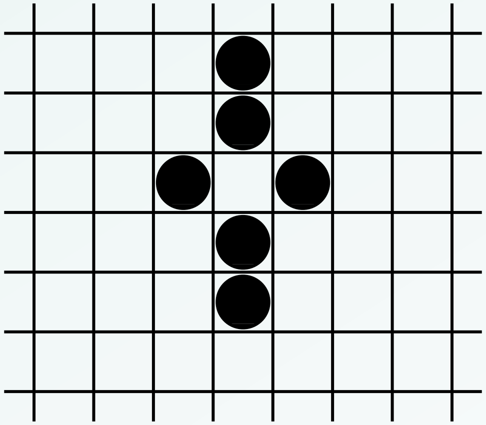
The local update rule asks each cell to check the present states of the eight surrounding cells.

- 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.

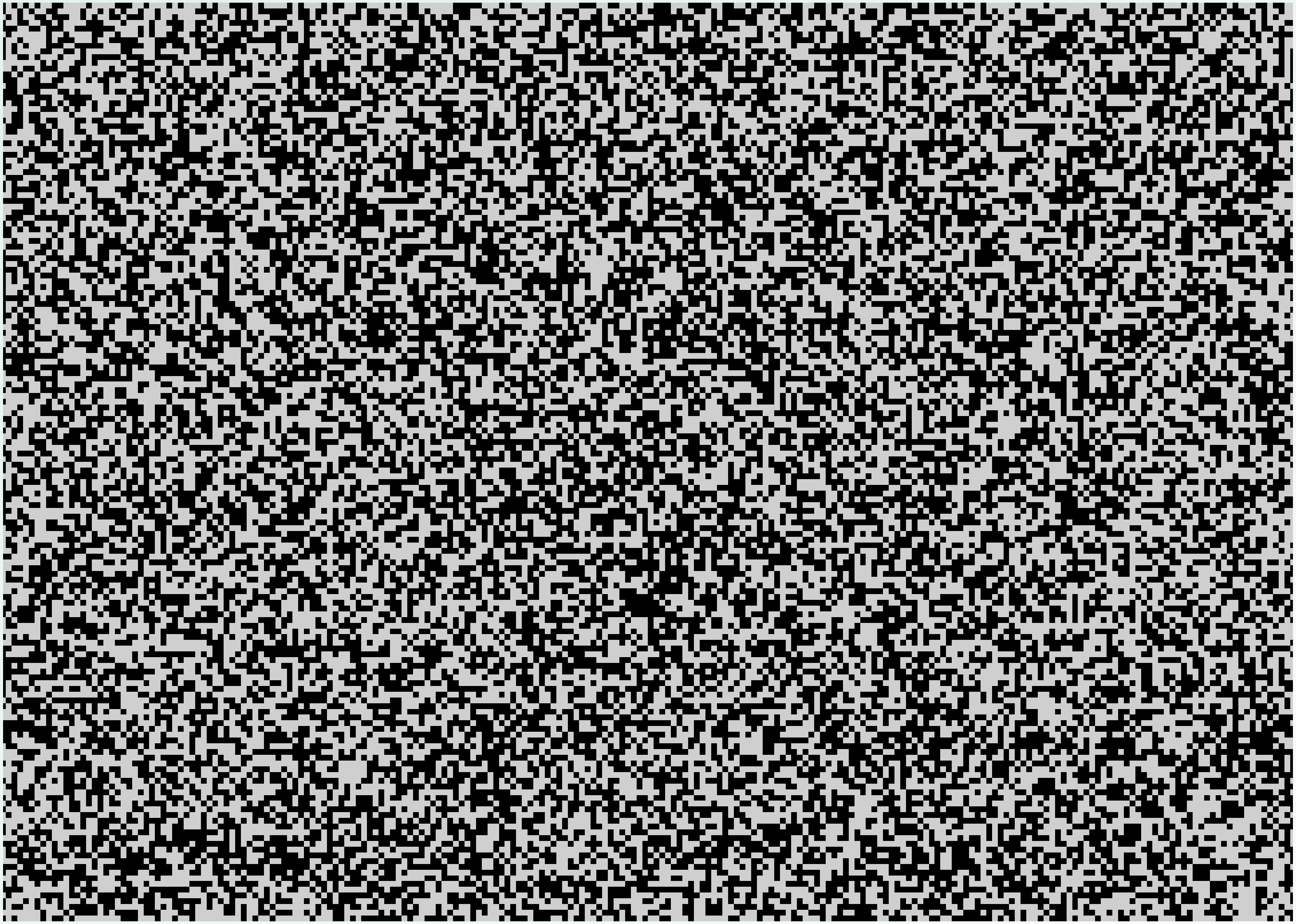


The local update rule asks each cell to check the present states of the eight surrounding cells.

- 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.

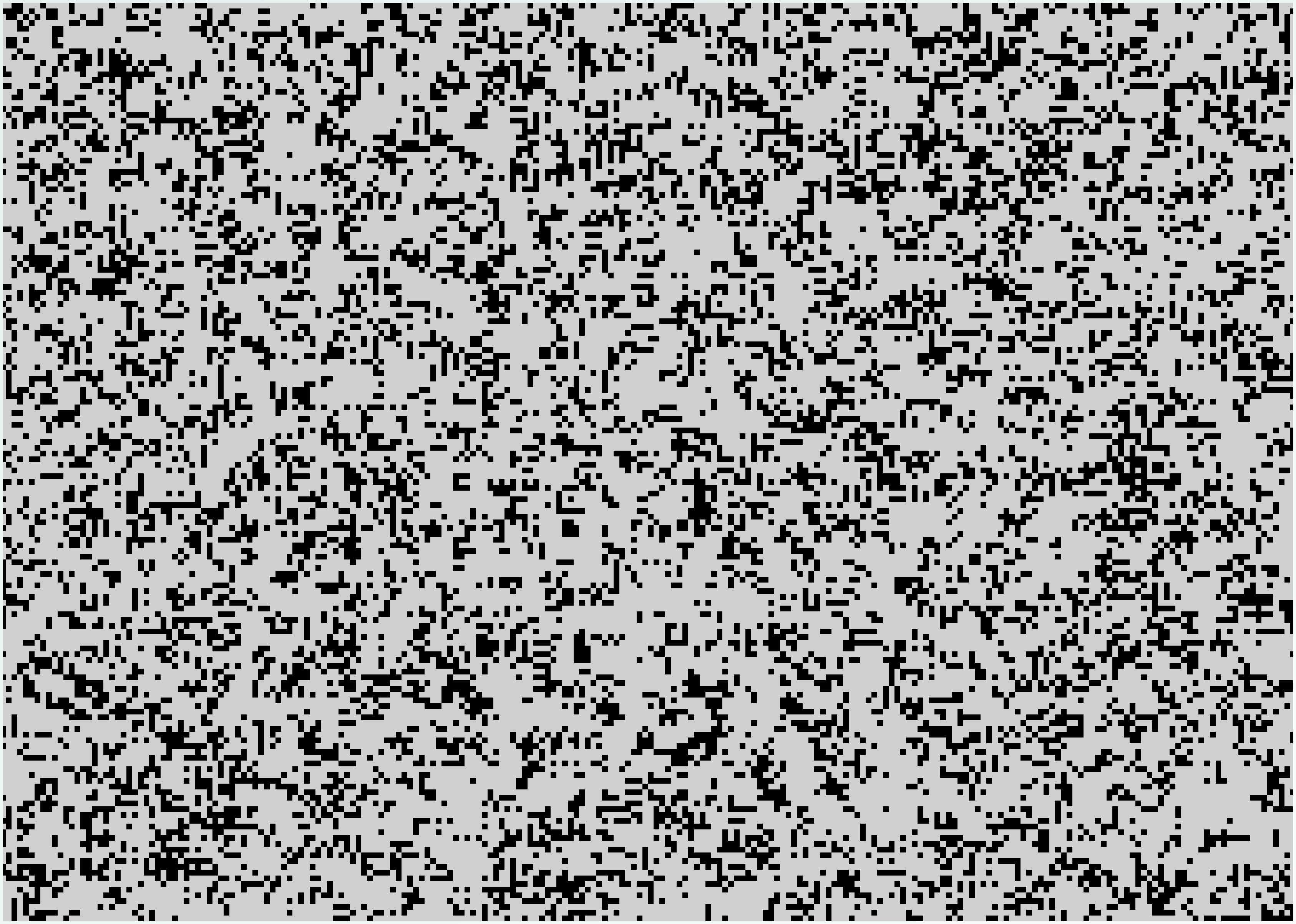


A typical snapshot of a time evolution in Game-of-Life:



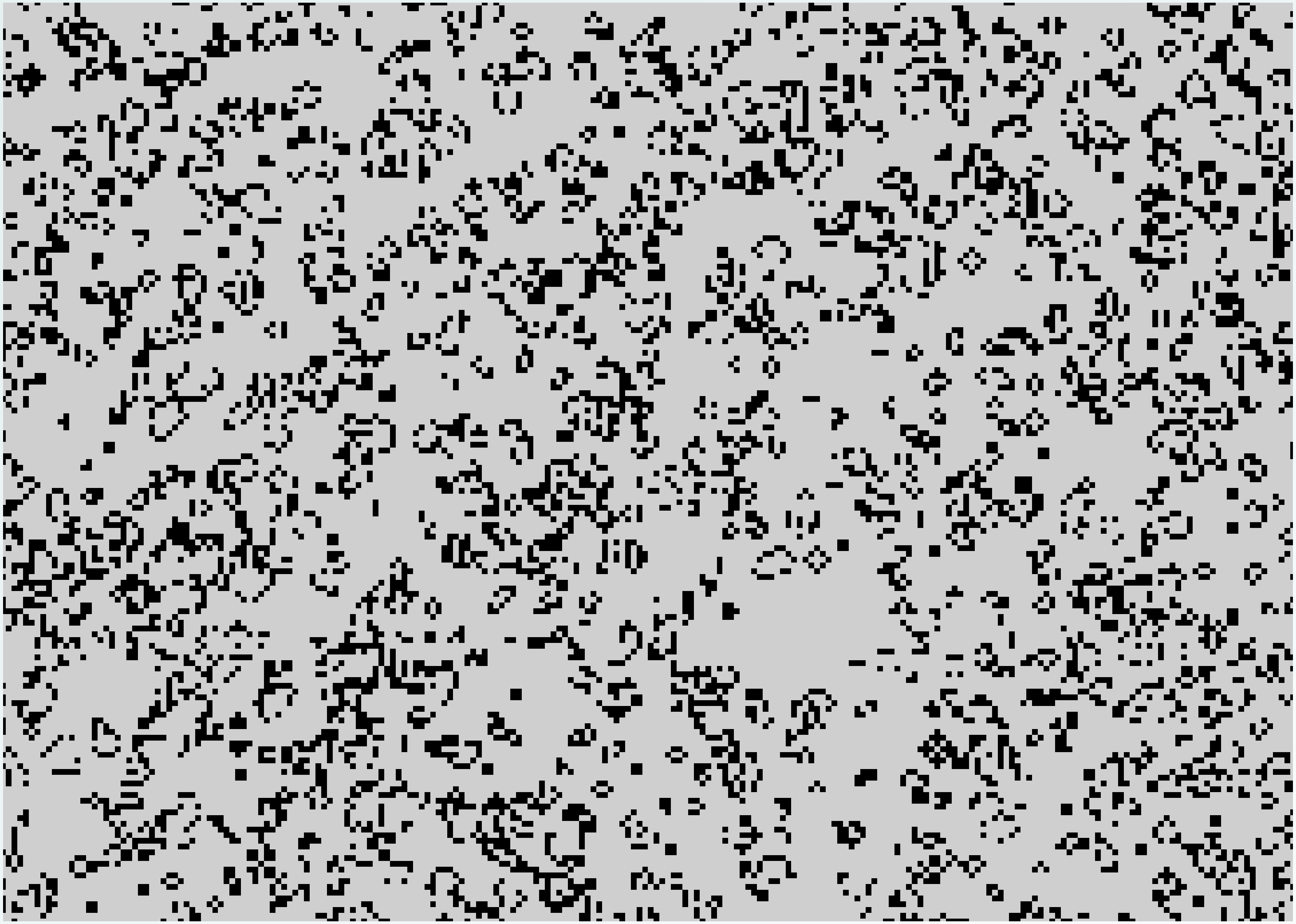
Initial uniformly random configuration.

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



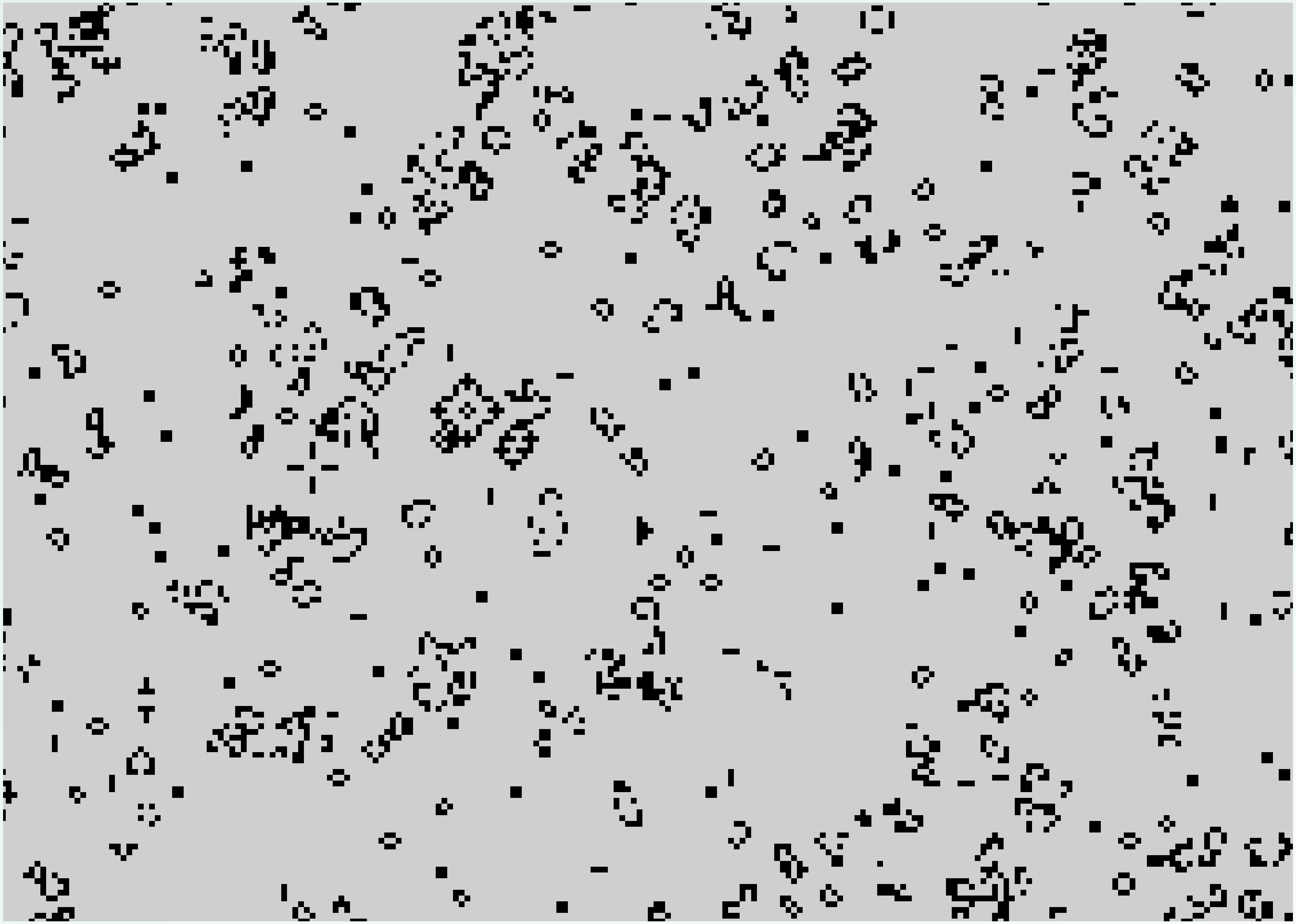
The next generation after all cells applied the update rule.

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



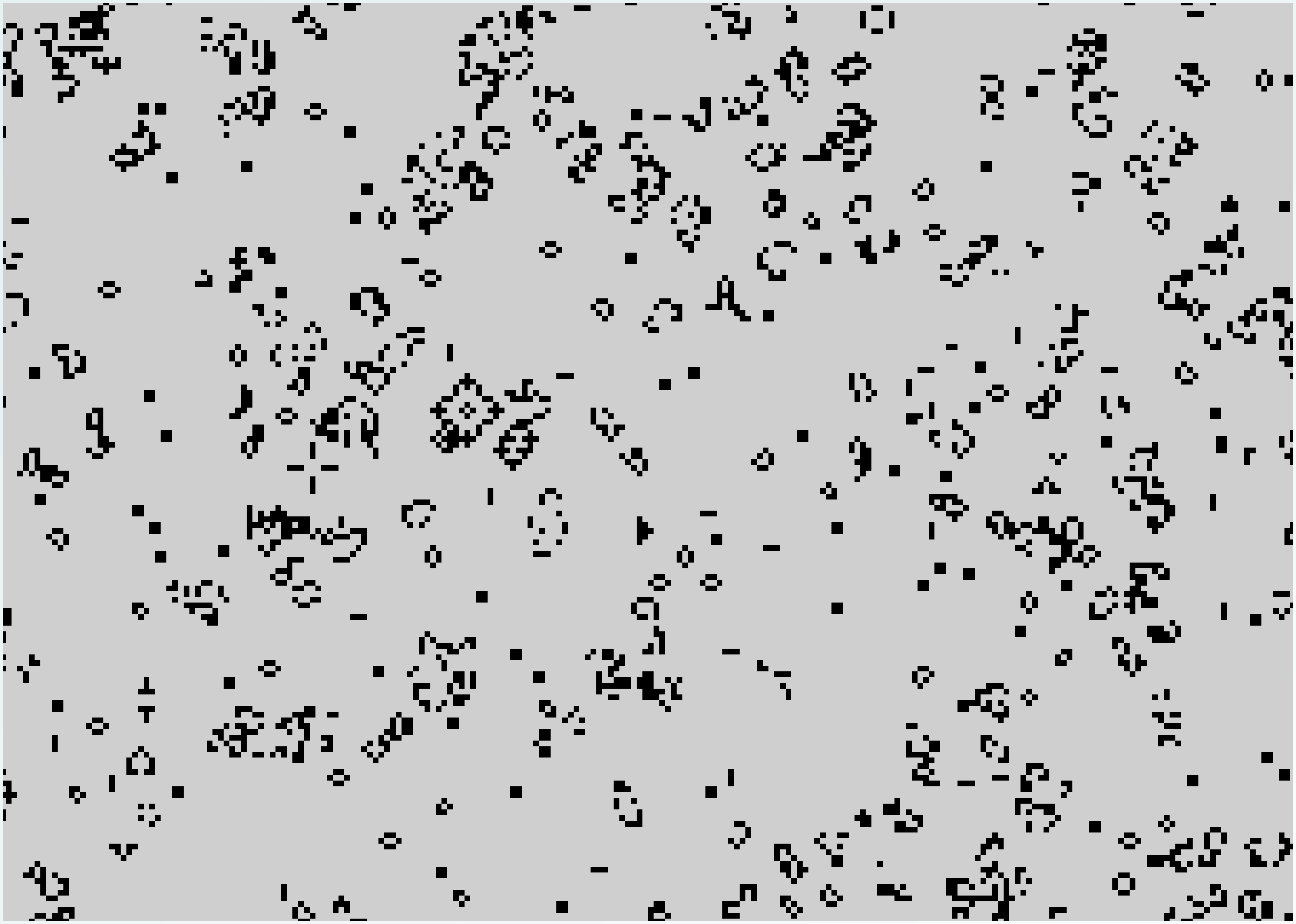
Generation 10

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.

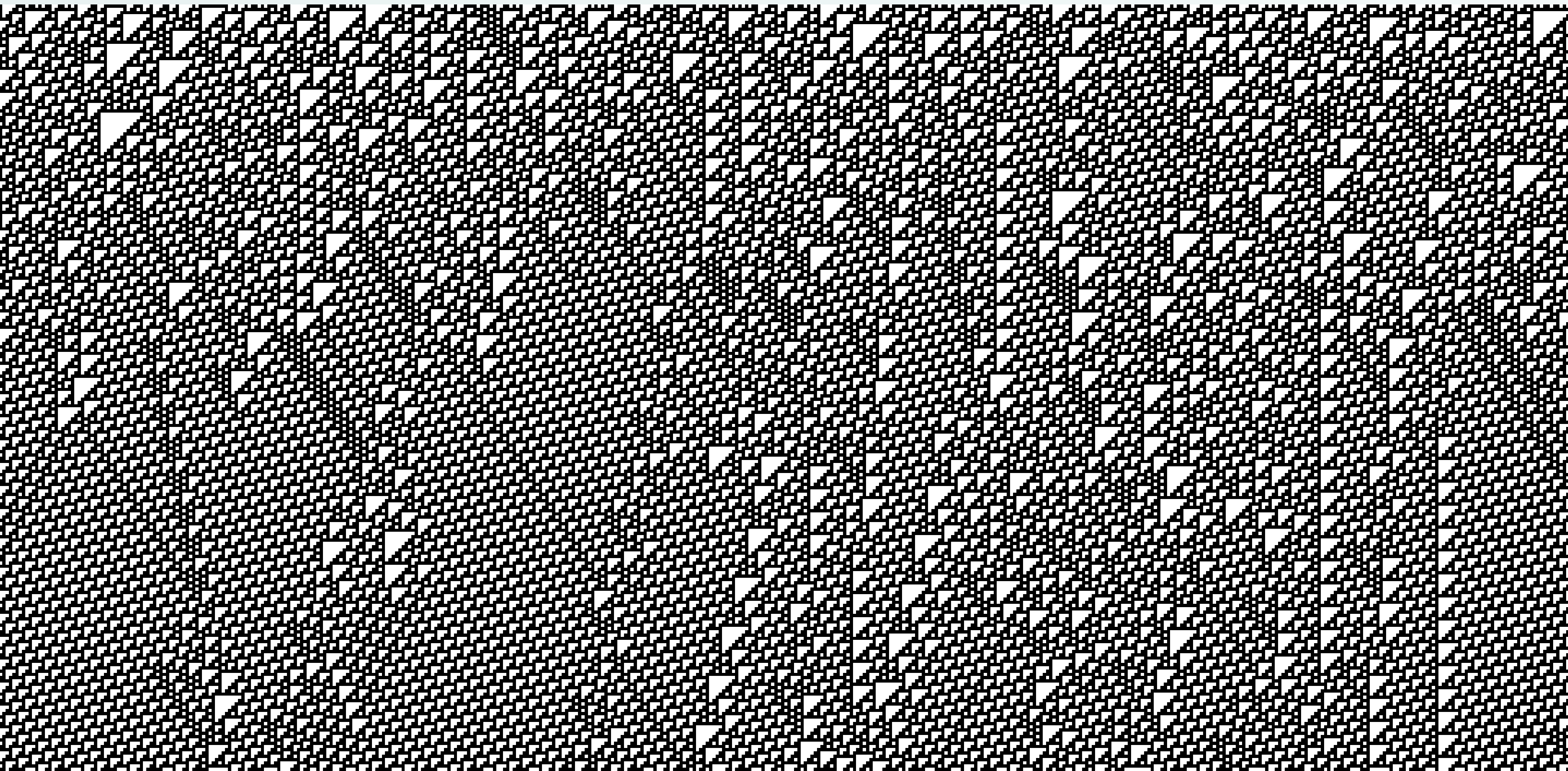
A one-dimensional CA with binary state set $\{0, 1\}$, i.e. a two-way infinite sequence of 0's and 1's.

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:



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,

- **Neighborhood** is a finite

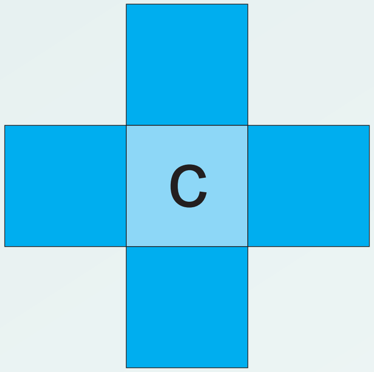
$$N \subseteq \mathbb{Z}^d$$

  that gives the offsets from each cell to its neighbors.

- The **neighbors** of a cell at location $\vec{x} \in \mathbb{Z}^d$ are the cells at locations

$$\vec{x} + \vec{n}, \text{ for } \vec{n} \in N.$$

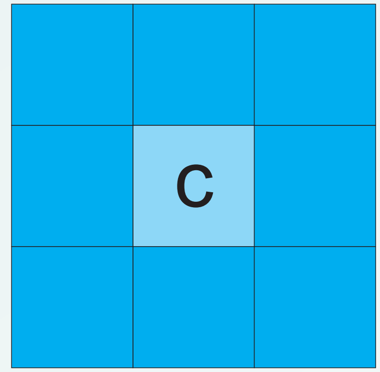
Typical two-dimensional neighborhoods:



Von Neumann

neighborhood

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



Moore

neighborhood

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

The **local rule**

$$f : S^N \longrightarrow S$$

gives the new state of a cell depending on the current pattern in its neighborhood.

The **local rule**

$$f : S^N \longrightarrow S$$

gives the new state of a cell depending on the current pattern in its neighborhood.

The local update rule f determines the **global dynamics**

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

that maps $c \mapsto G(c)$ where

$$\forall \vec{x} \in \mathbb{Z}^d \qquad G(c)_{\vec{x}} = f(c_{\vec{x}+N}).$$

The **local rule**

$$f : S^N \longrightarrow S$$

gives the new state of a cell depending on the current pattern in its neighborhood.

The local update rule f determines the **global dynamics**

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

that maps $c \mapsto G(c)$ where

$$\forall \vec{x} \in \mathbb{Z}^d \qquad G(c)_{\vec{x}} = f(c_{\vec{x}+N}).$$

Function G is our main object of study and we simply call it a **CA function.** In algorithmic questions we use its finite presentation (the local rule).

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:

The metric induces a **compact** topology on $S^{\mathbb{Z}^d}$.

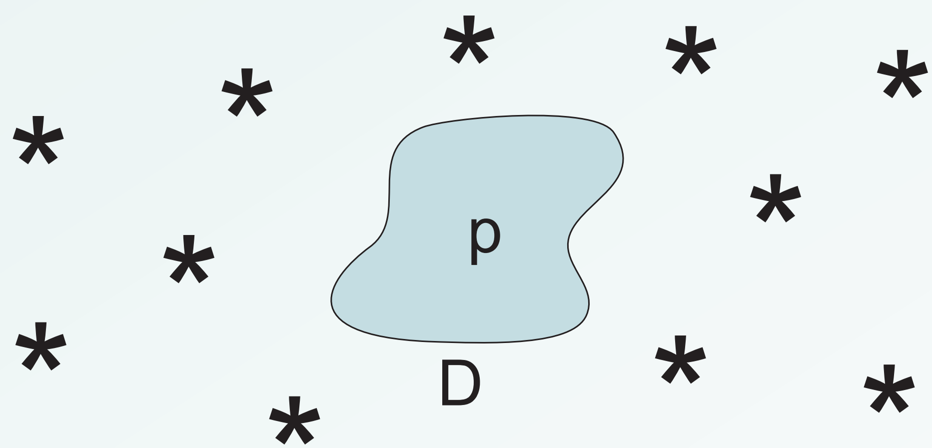
The topology is generated by the **cylinder sets**.

A **finite pattern** is an assignment $p : D \longrightarrow S$ of states in a finite domain $D \subset \mathbb{Z}^d$.

The **cylinder** determined by D, p is the set

$$\{c \in S^{\mathbb{Z}^d} \mid \forall \vec{x} \in D \ : \ c_{\vec{x}} = p_{\vec{x}}\}$$

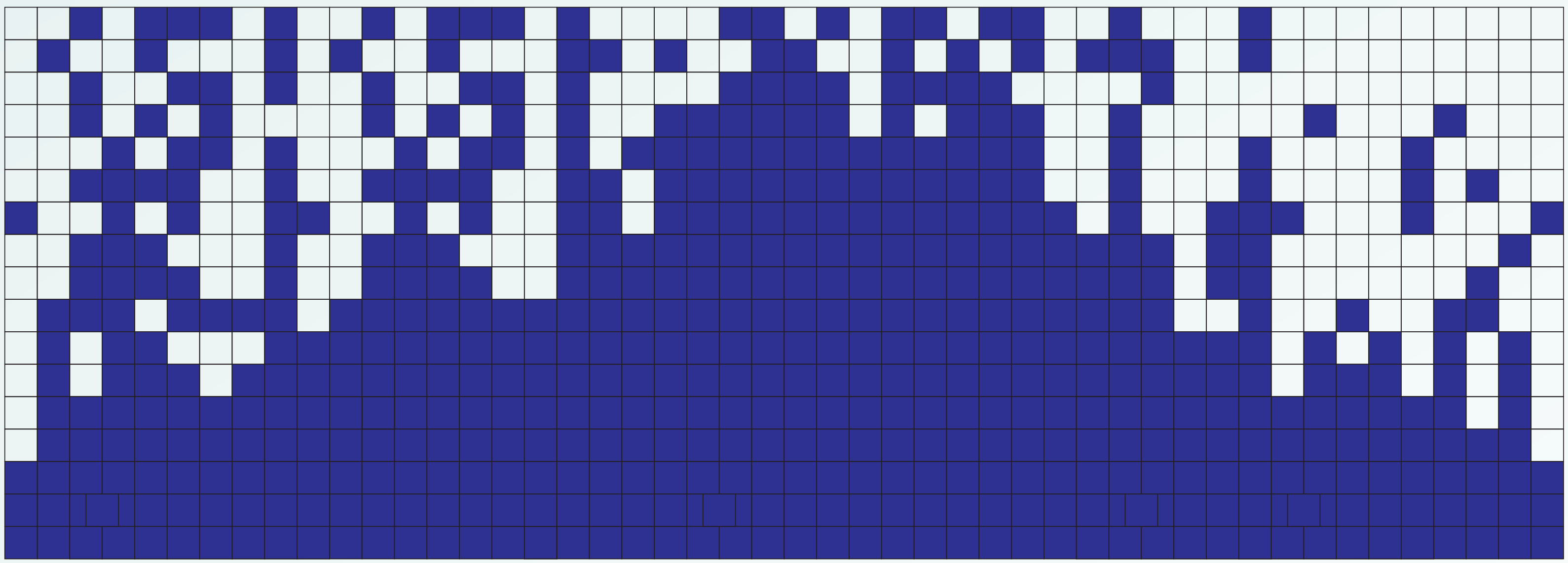
of all configurations that agree with p in domain D.



Cylinders are both open and closed: They form a **clopen basis** of the topology.

Under this topology, a sequence $c_1, c_2, \ldots$ of configurations **converges** to $c \in S^{\mathbb{Z}^d}$ if and only if for all cells $\vec{x} \in \mathbb{Z}^d$ and for all sufficiently large i holds

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



Compactness of the topology means that all infinite sequences $c_1, c_2, \ldots$ of configurations have converging subsequences.

All cellular automata are **continuous** transformations

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

under the topology.

All cellular automata are **continuous** transformations

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

under the topology.

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.

Some symbolic dynamics terminology:

- The set $S^{\mathbb{Z}^d}$, together with the shift maps, is the d-dimensional **full shift**.
- Topologically closed, shift invariant subsets of $S^{\mathbb{Z}^d}$ are called **subshifts**.
- Cellular automata are the endomorphisms of the full shift.

Finite and 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 **finite configurations** or on **periodic configurations**.

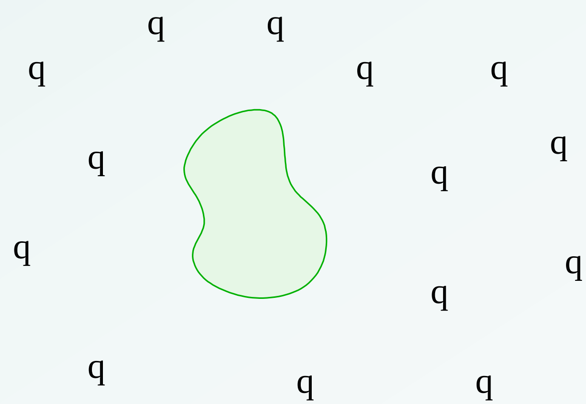
Finite configurations: One state $q \in S$ is often identified as the **quiescent** state, and it is expected to be stable:

$$f(q, q, \ldots, q) = q.$$

A configuration $c \in S^{\mathbb{Z}^d}$ is called **finite** if the set

$$\{\vec{n} \in \mathbb{Z}^d \mid c(\vec{n}) \neq q\}$$

is finite.



Due to stability of q, CA G maps finite configurations to finite 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 $\vec{r}$:

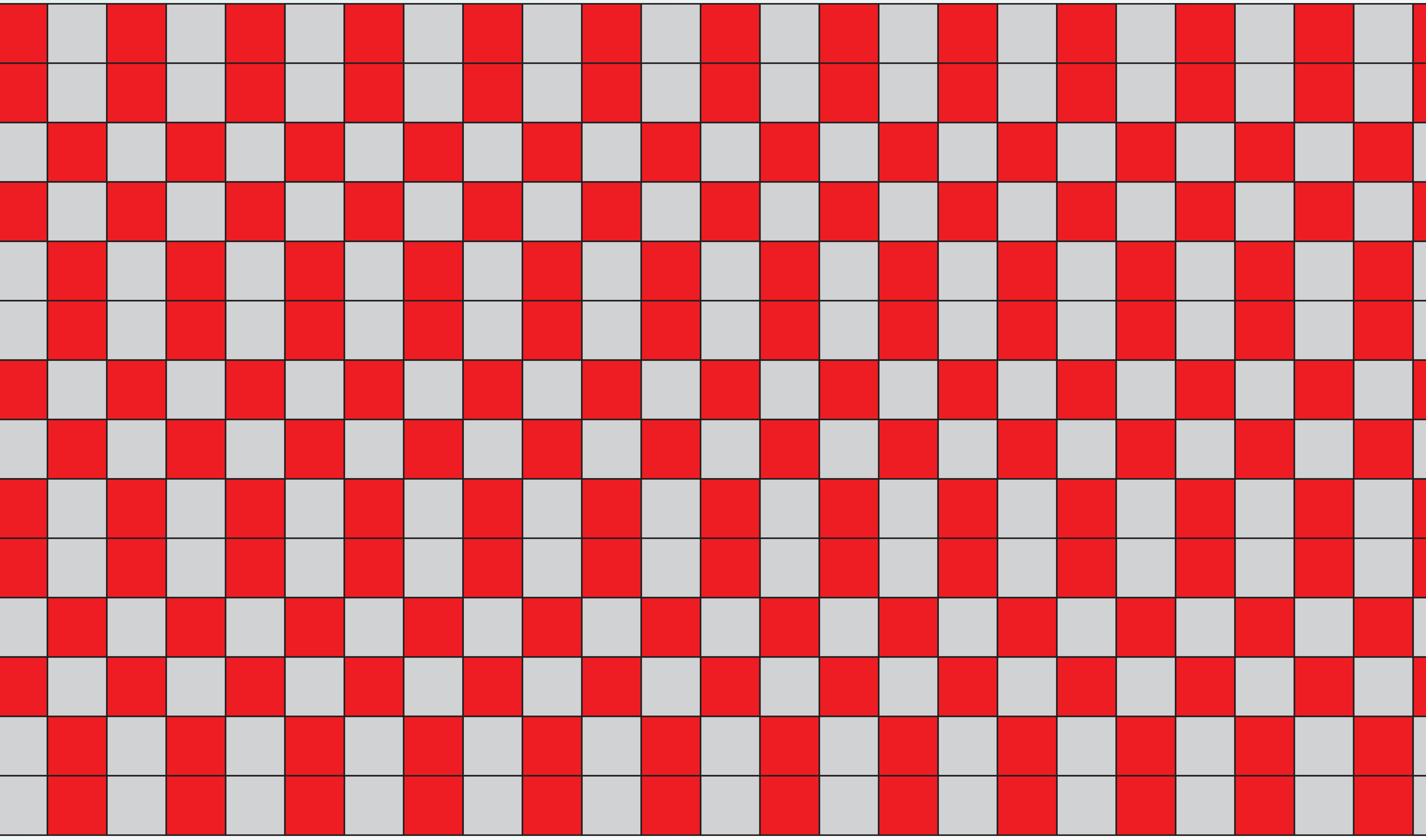
$$\tau(c) = c.$$

CA functions commute with translations, so we also have

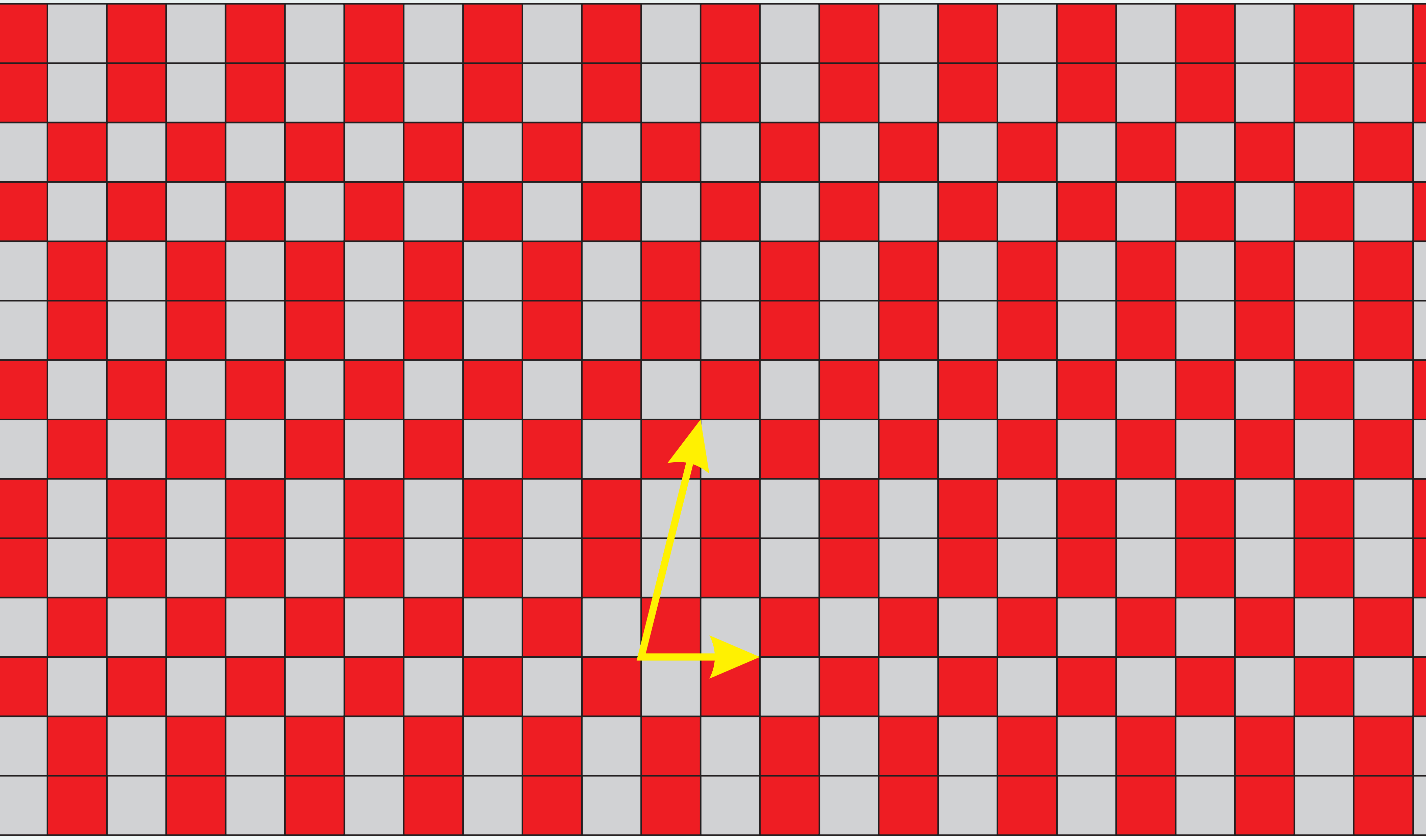
$$\tau(G(c)) = G(\tau(c)) = G(c).$$

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

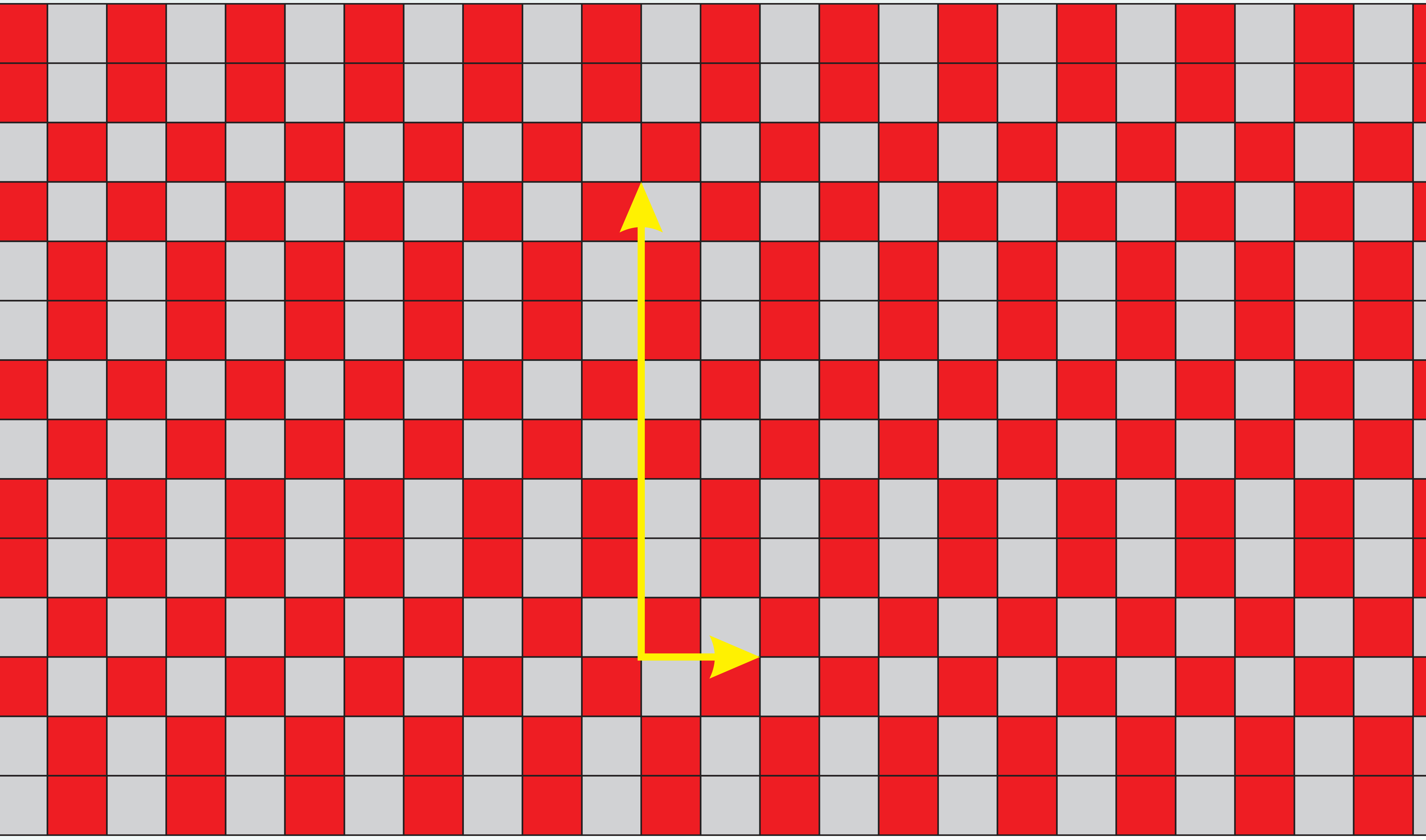
Configuration $c \in S^{\mathbb{Z}^d}$ is (fully) **periodic** if it has d linearly independent periods.



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.

Finite and periodic configurations allow simulations of cellular automata on finitely presented configurations. The use of periodic configurations is usually termed **periodic boundary conditions**.

Finite configurations and periodic configurations are **dense** in $S^{\mathbb{Z}^d}$: each cylinder contains finite and periodic configurations.

Reversible CA

A CA is called

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

Reversible CA

A CA is called

- **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.

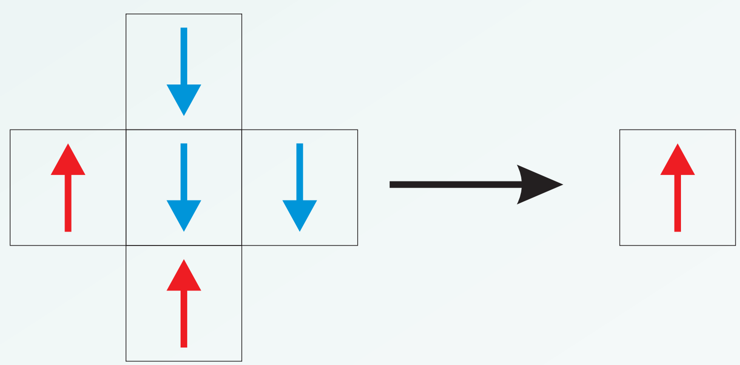
$$G \circ F = F \circ G = \text{identity function.}$$

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

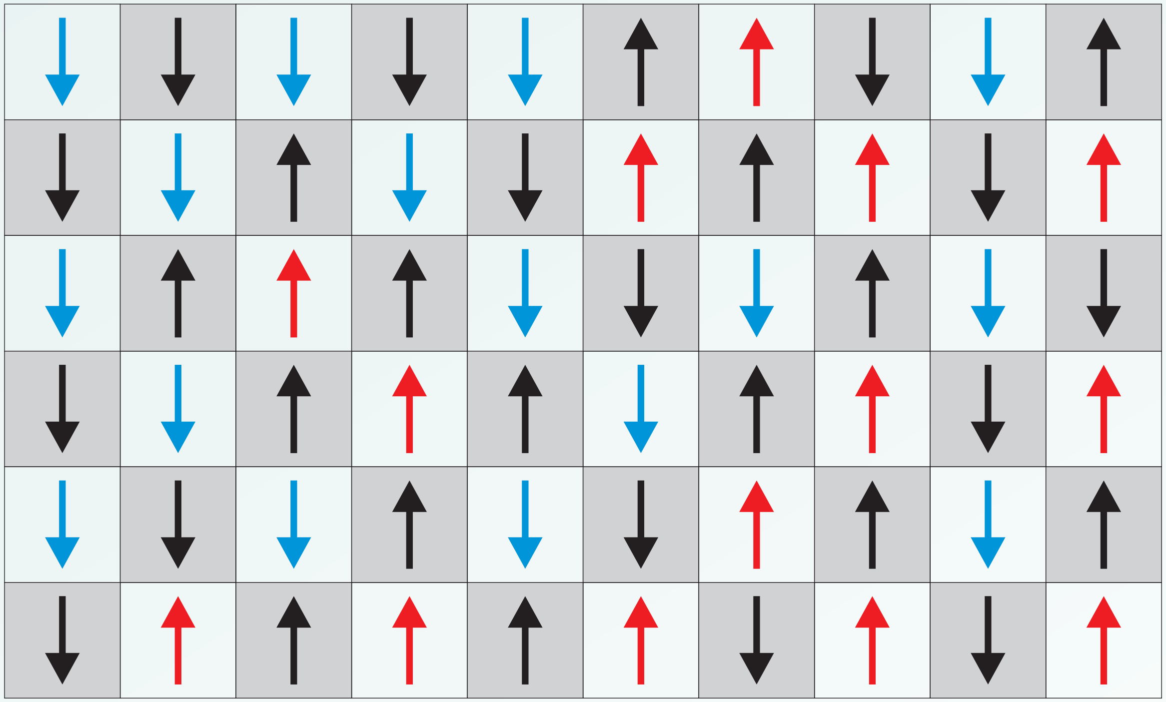
Game-of-Life and Rule 110 are irreversible: Configurations may have several pre-images.

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

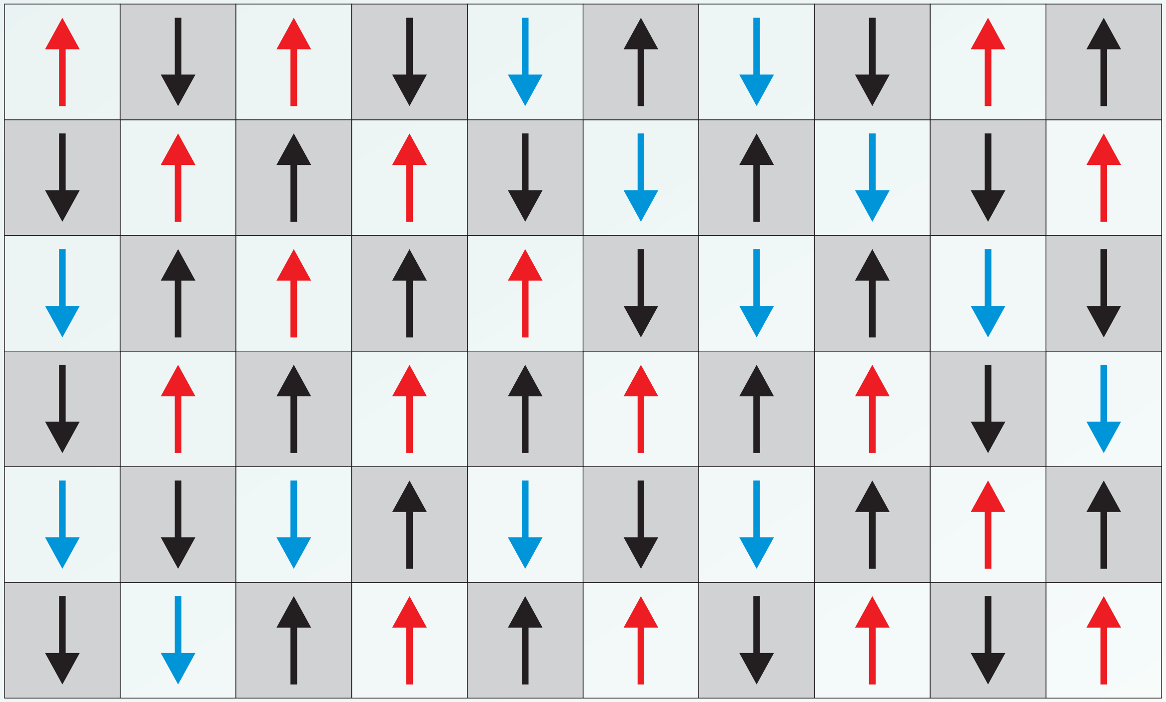
Each cell has a spin that is directed either up or down. The direction of a spin is swapped if and only if among the four immediate neighbors there are exactly two cells with spin up and two cells with spin down:



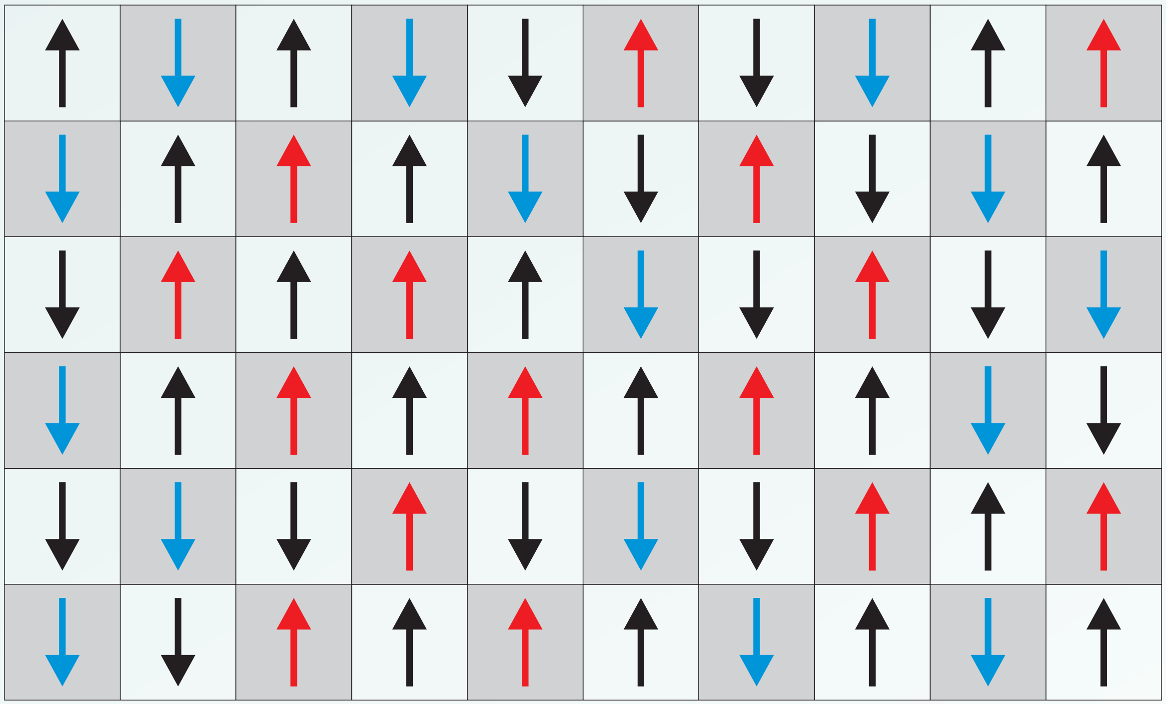
The twist that makes the Q2R rule reversible: Color the space as a checker-board. On even time steps only update the spins of the white cells and on odd time steps update the spins of the black cells.



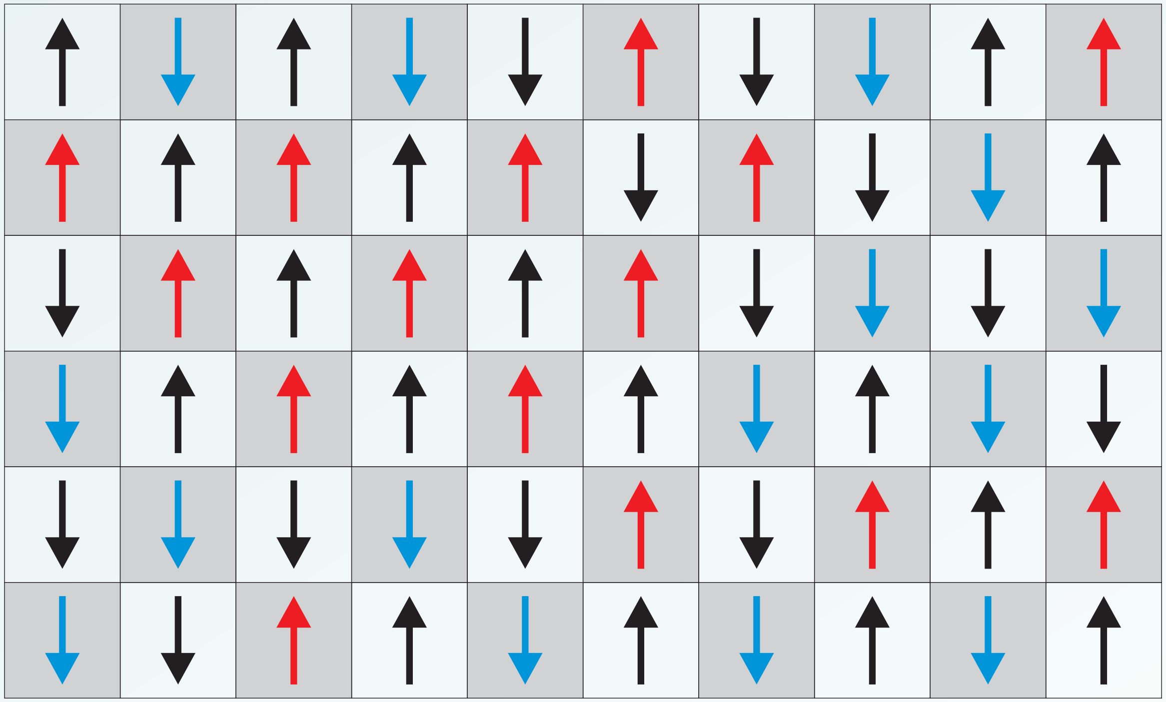
The twist that makes the Q2R rule reversible: Color the space as a checker-board. On even time steps only update the spins of the white cells and on odd time steps update the spins of the black cells.



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.

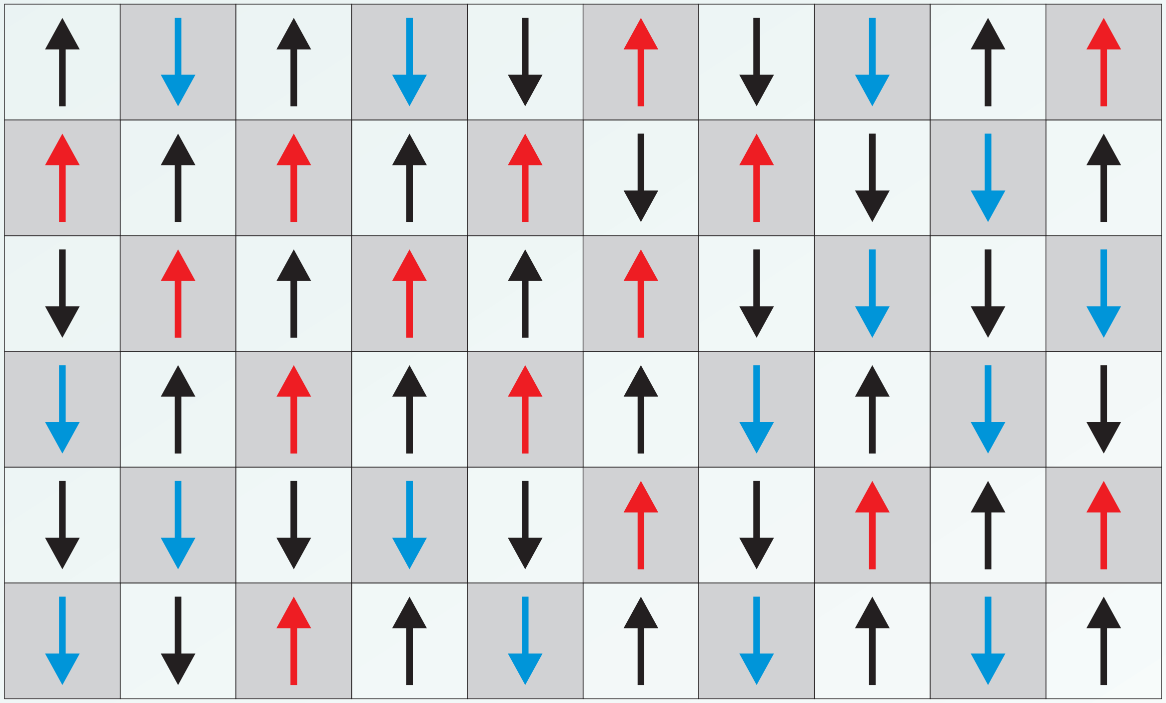


The twist that makes the Q2R rule reversible: Color the space as a checker-board. On even time steps only update the spins of the white cells and on odd time steps update the spins of the black cells.

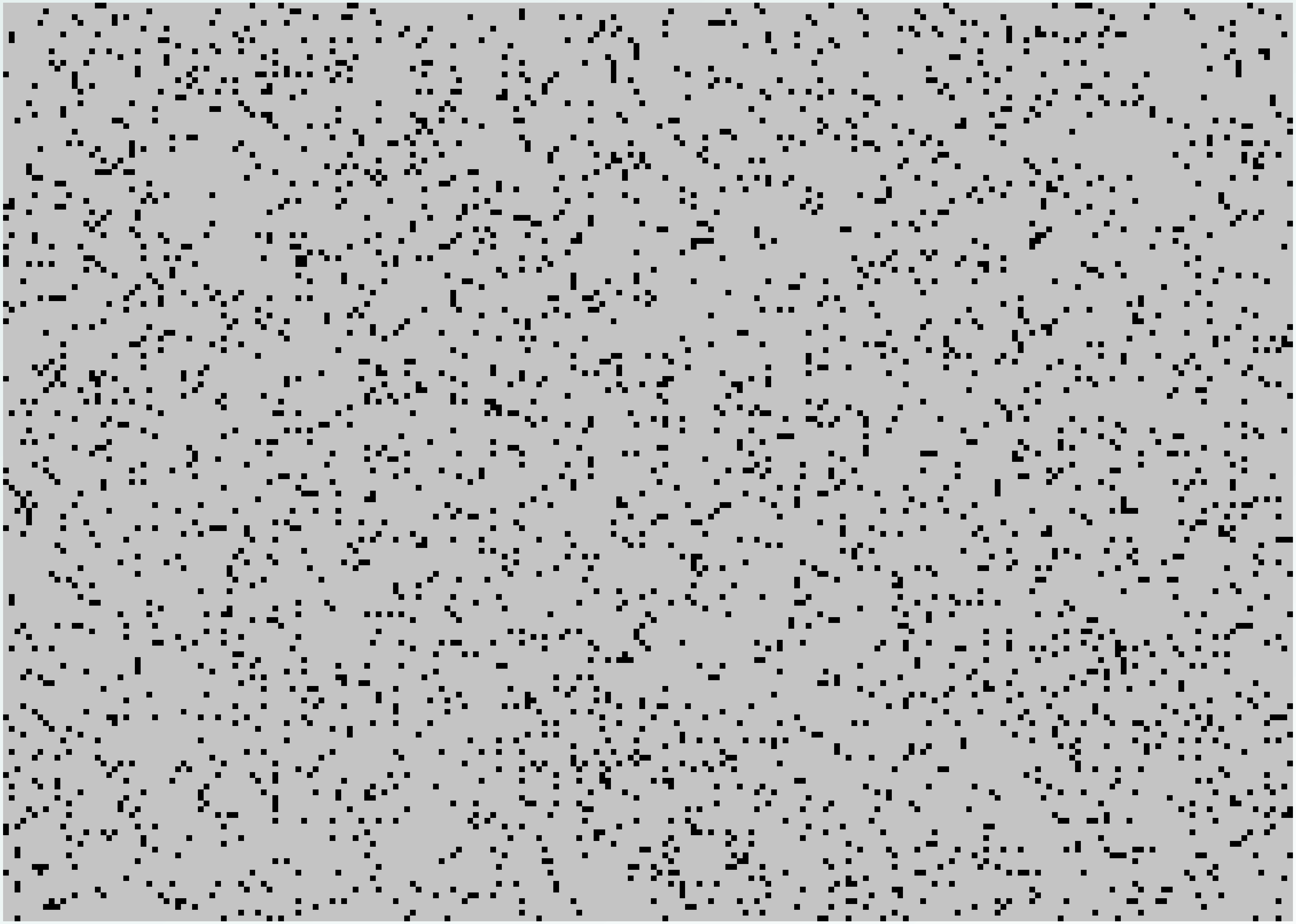


Q2R is **reversible**: The same rule (applied again on squares of the same color) reconstructs the previous generation.

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

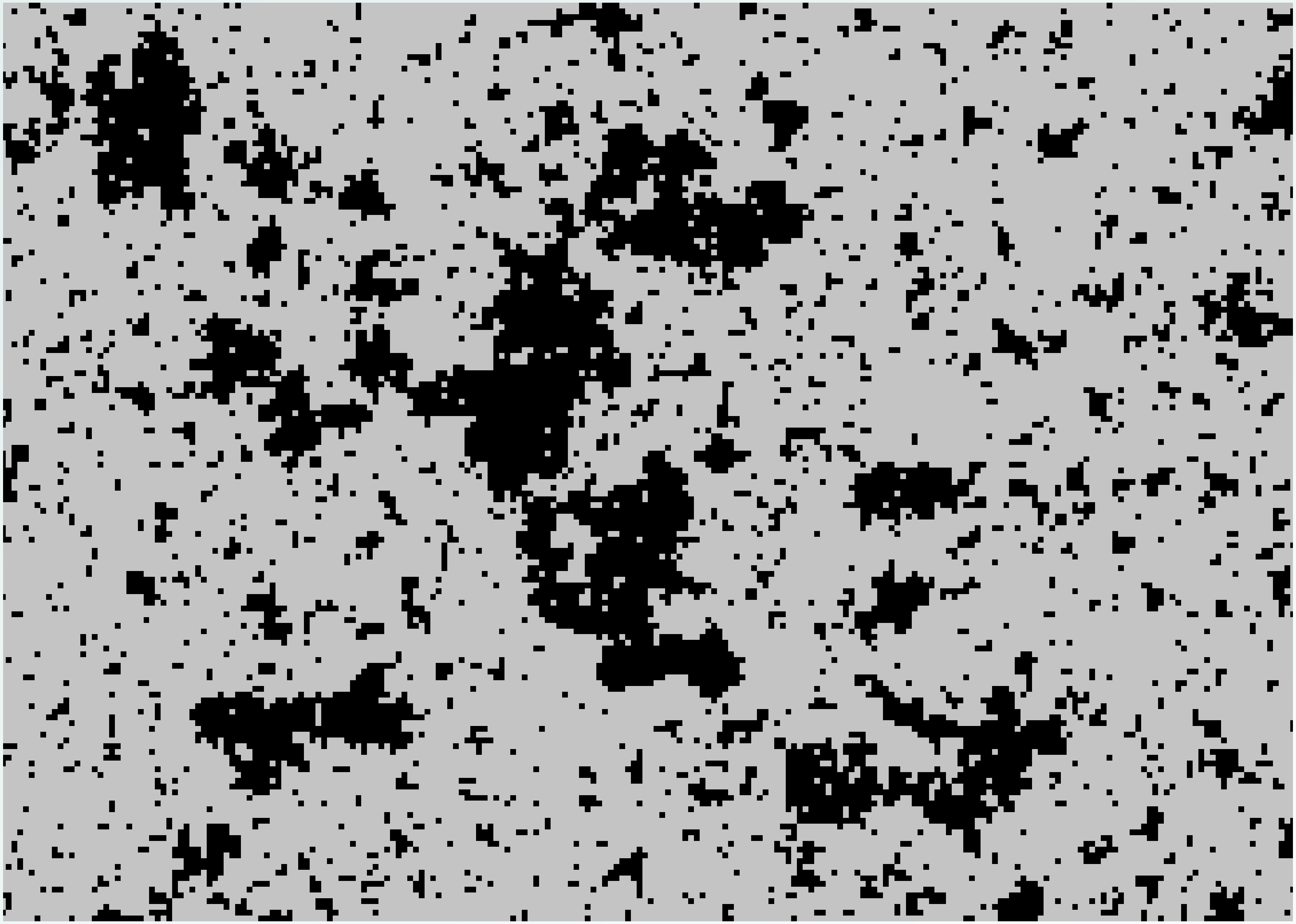


Evolution of Q2R from an uneven random distribution of spins:



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.

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. $\square$

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.

Universality in reversible CA (RCA)

Simulating a Turing machine by an irreversible CA is trivial.

But computation universality is possible also under the reversibility constraint:

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

Universality in reversible CA (RCA)

Simulating a Turing machine by an irreversible CA is trivial.

But computation universality is possible also under the reversibility constraint:

- **T. Toffoli (1977)**. Any d-dimensional CA can be simulated by a $(d+1)$-dimensional RCA.
- **K. Morita and M. Harao (1989)**. Any reversible Turing machine can be simulated by one-dimensional RCA.

Universality in reversible CA (RCA)

Simulating a Turing machine by an irreversible CA is trivial.

But computation universality is possible also under the reversibility constraint:

- **T. Toffoli (1977)**. Any d-dimensional CA can be simulated by a $(d + 1)$-dimensional RCA.
- **K. Morita and M. Harao (1989)**. Any reversible Turing machine can be simulated by one-dimensional RCA.
- **J.-C. Dubacq (1995)**. Any Turing machine can be simulated in real time by a one-dimensional RCA.

The proofs use partitioned CA, a technique to guarantee reversibility.

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

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

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

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

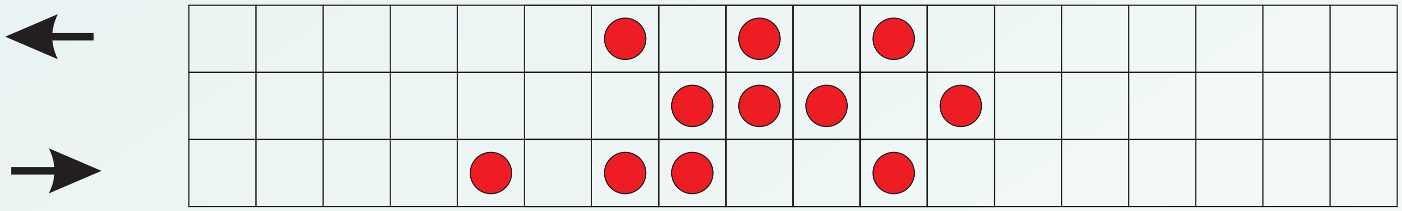
$$\pi : S \longrightarrow S$$

  applied in each cell independent of its neighbors.

The two phases **alternate**.

Example: PCA with three binary tracks:

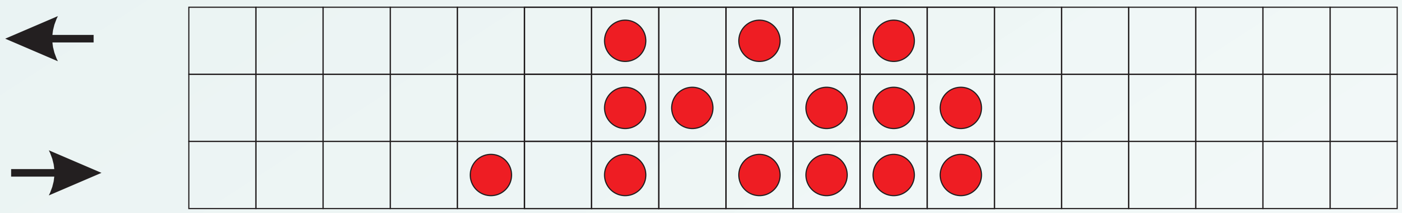
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \mapsto \begin{bmatrix} a \\ a+b \pmod 2 \\ b+c \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

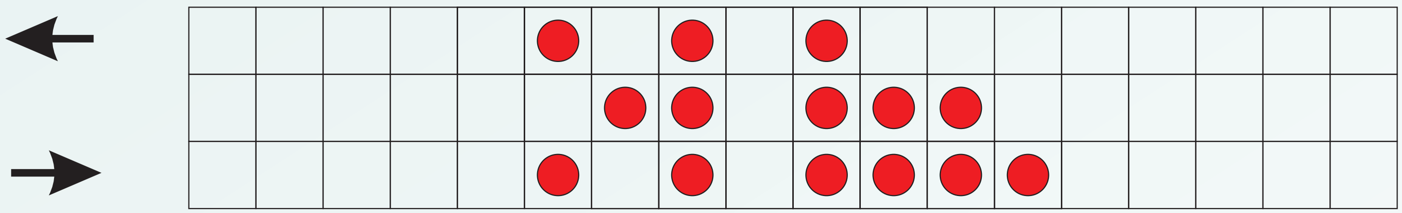
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \mapsto \begin{bmatrix} a \\ a+b \pmod 2 \\ b+c \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

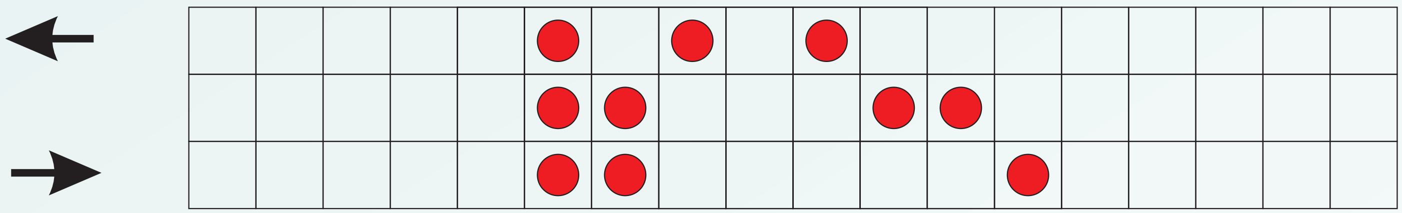
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \mapsto \begin{bmatrix} a \\ a+b \pmod 2 \\ b+c \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

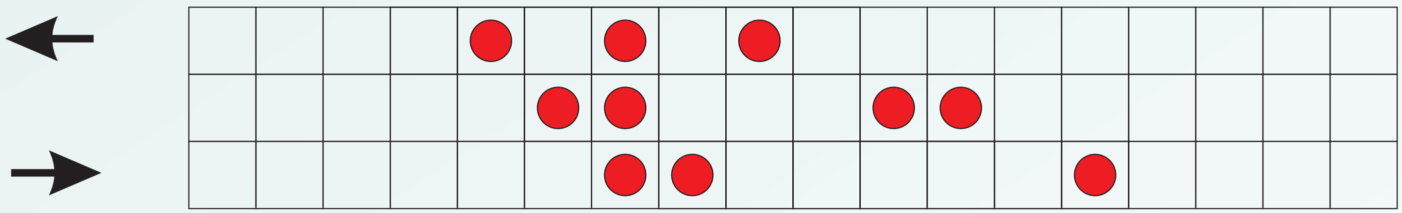
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \quad \mapsto \quad \begin{bmatrix} a \\ a+b \pmod 2 \\ b+c \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

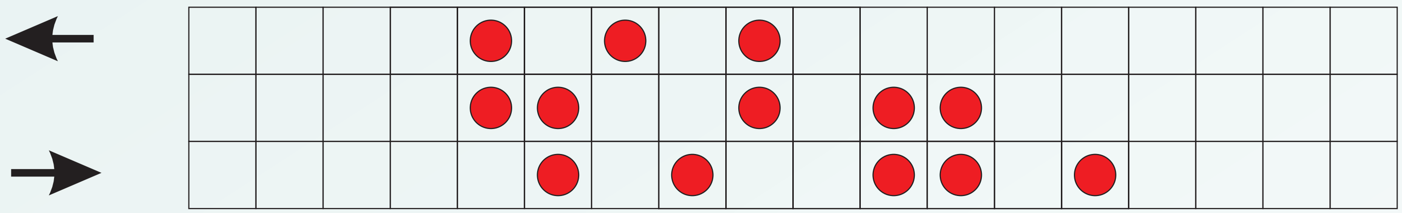
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \mapsto \begin{bmatrix} a \\ a+b \pmod 2 \\ b+c \pmod 2 \end{bmatrix}$$



PCA are reversible as both elementary steps are reversible.

Example: PCA with three binary tracks:

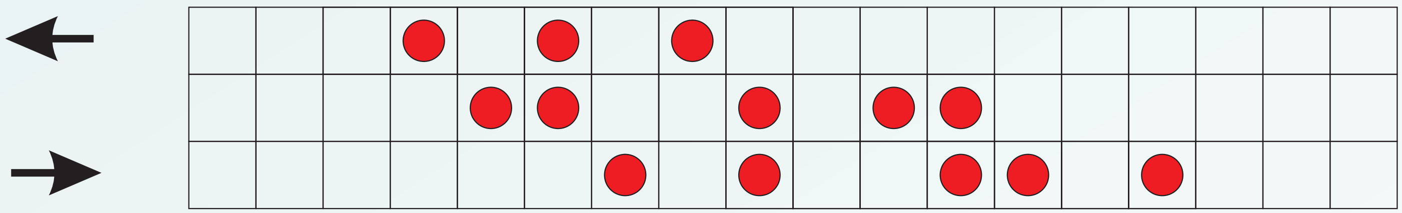
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \mapsto \begin{bmatrix} a \\ a+b \pmod 2 \\ b+c \pmod 2 \end{bmatrix}$$



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.

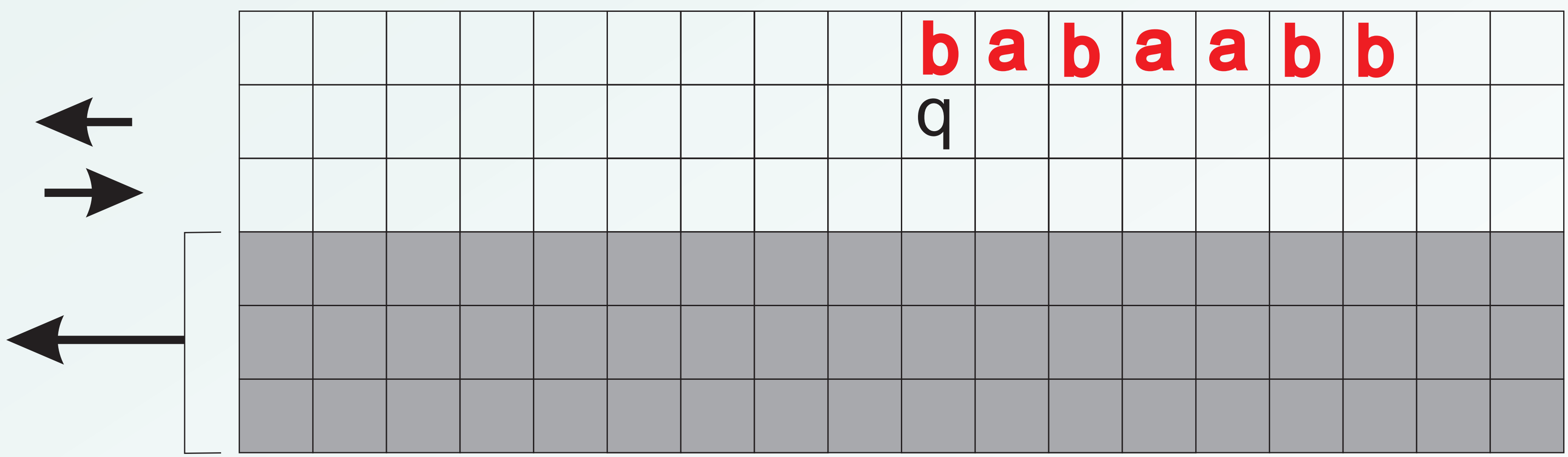
Track (2) or (3) stores the Turing machine state. They are shifted one cell left and right respectively. The state is stored on the track which moves to the direction indicated by the TM instruction being executed.

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.

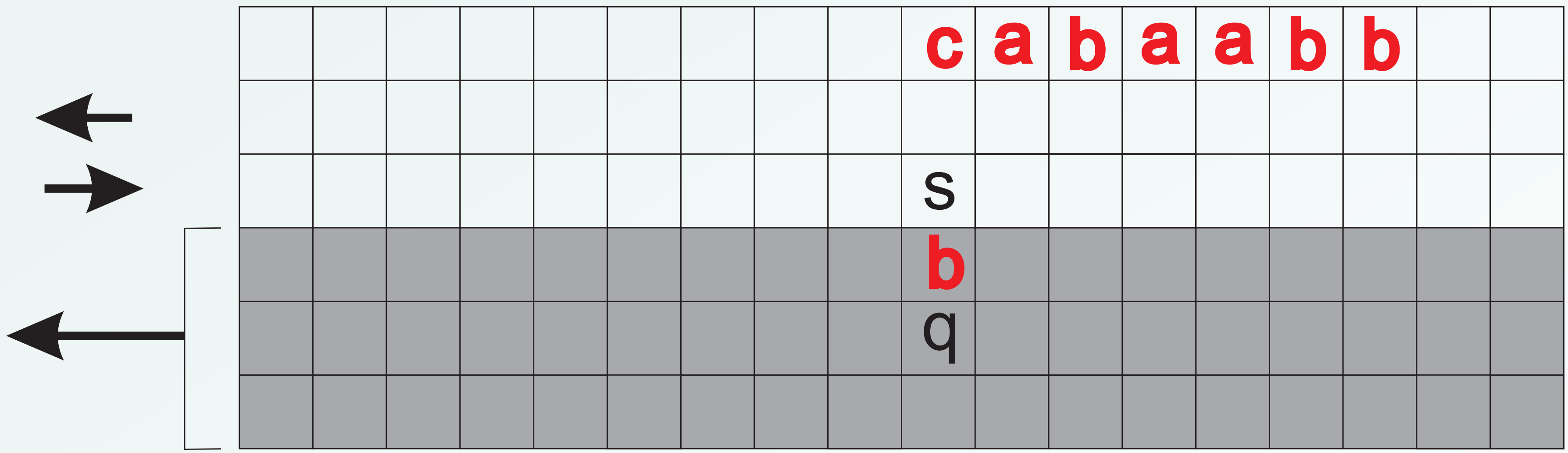
- updates the first three tracks according to the TM instruction.



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

The permutation π

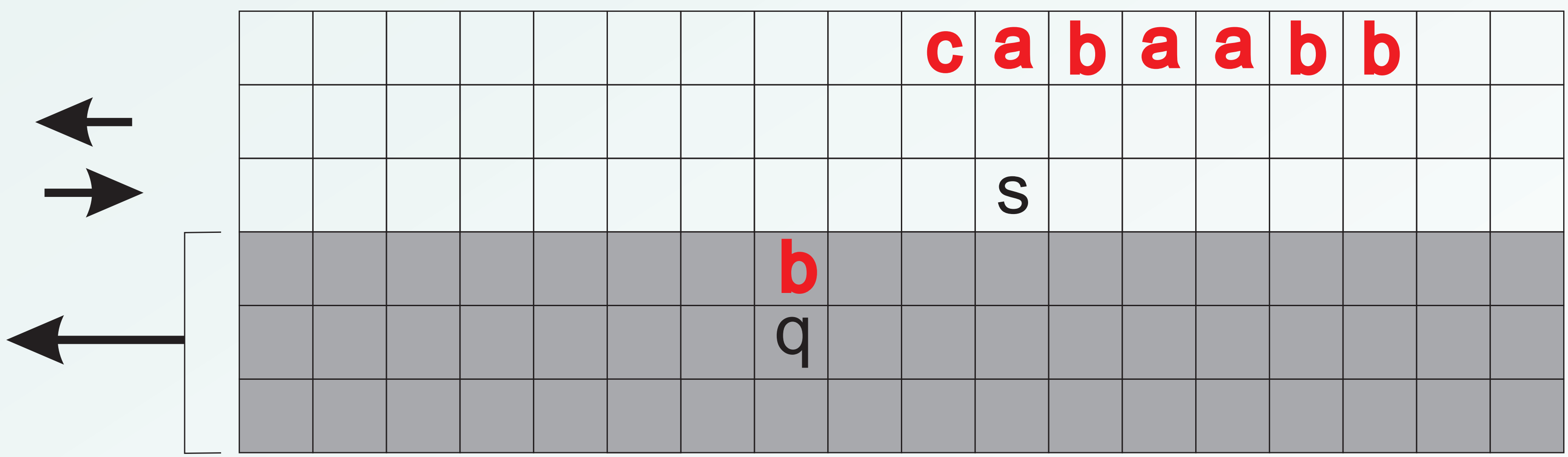
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.
- updates the first three tracks according to the TM instruction.



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.

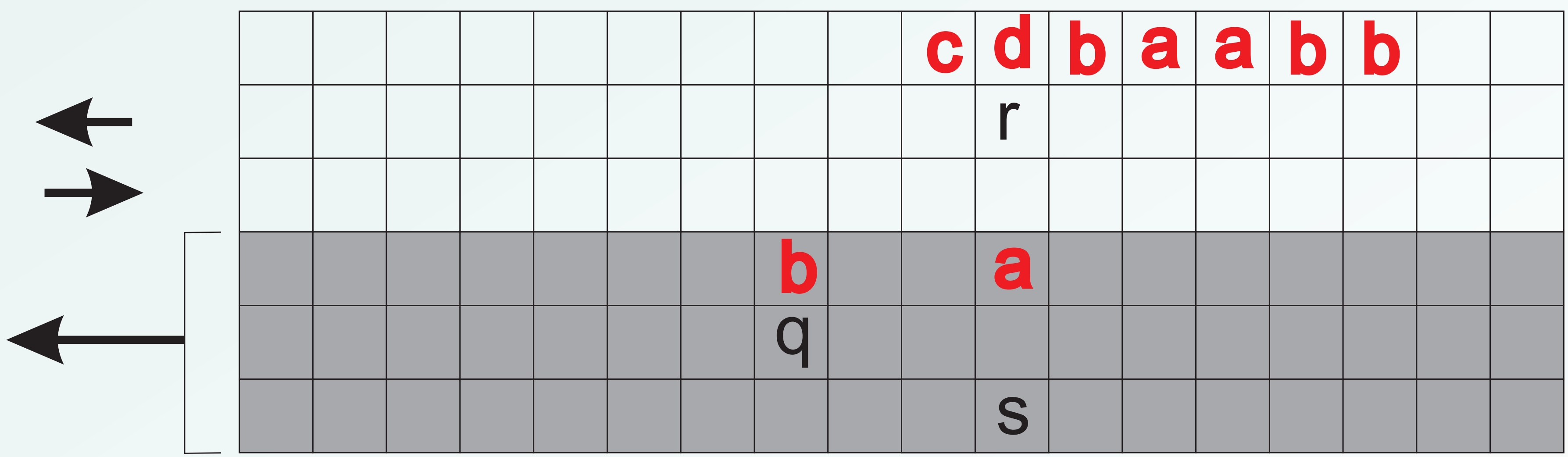


TM(s,a) = (r,d, $\leftarrow$)

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.

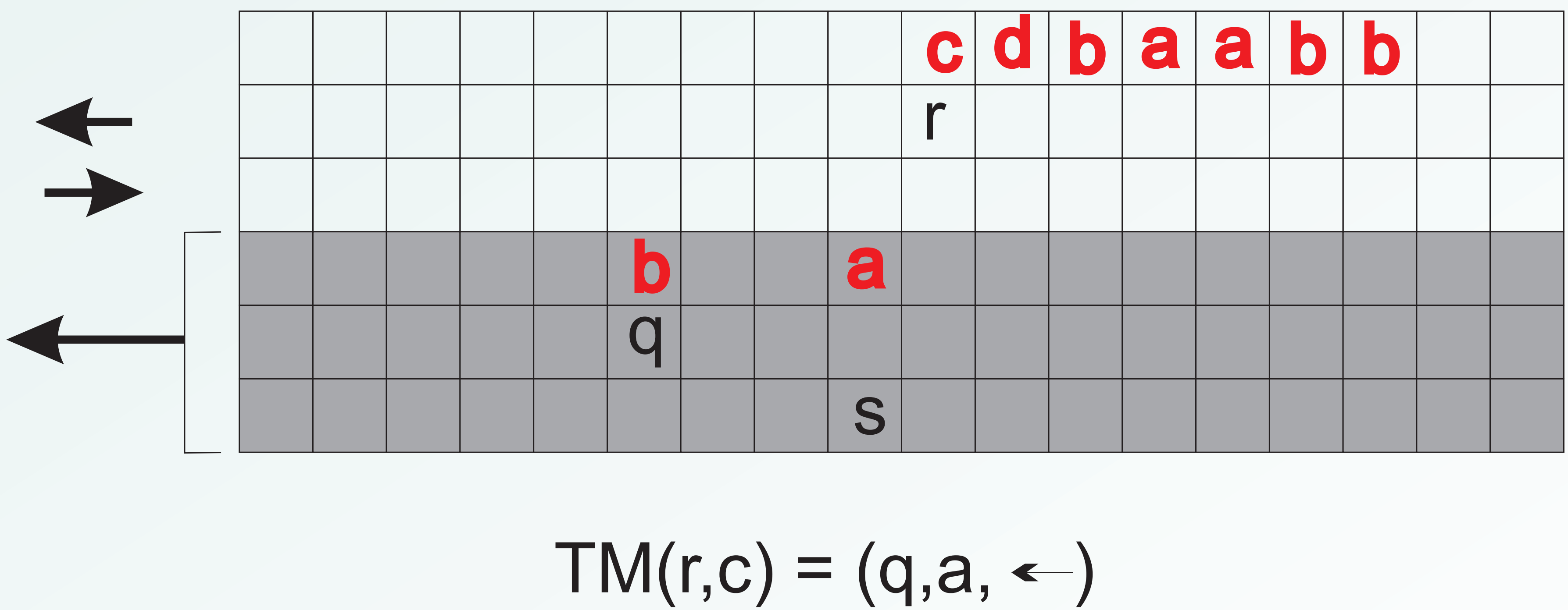


TM(s,a) = (r,d, $\leftarrow$)

The permutation π

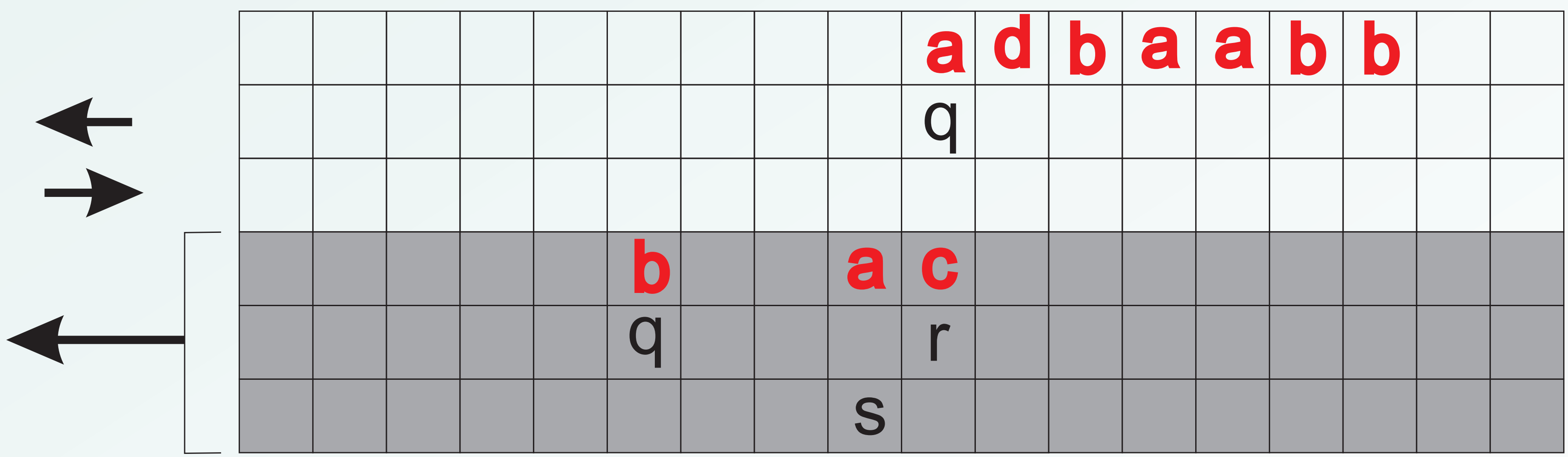
- copies the contents of tracks (1),(2),(3) on the garbage track. The copying is done only at the cell containing the TM.

- updates the first three tracks according to the TM instruction.



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.

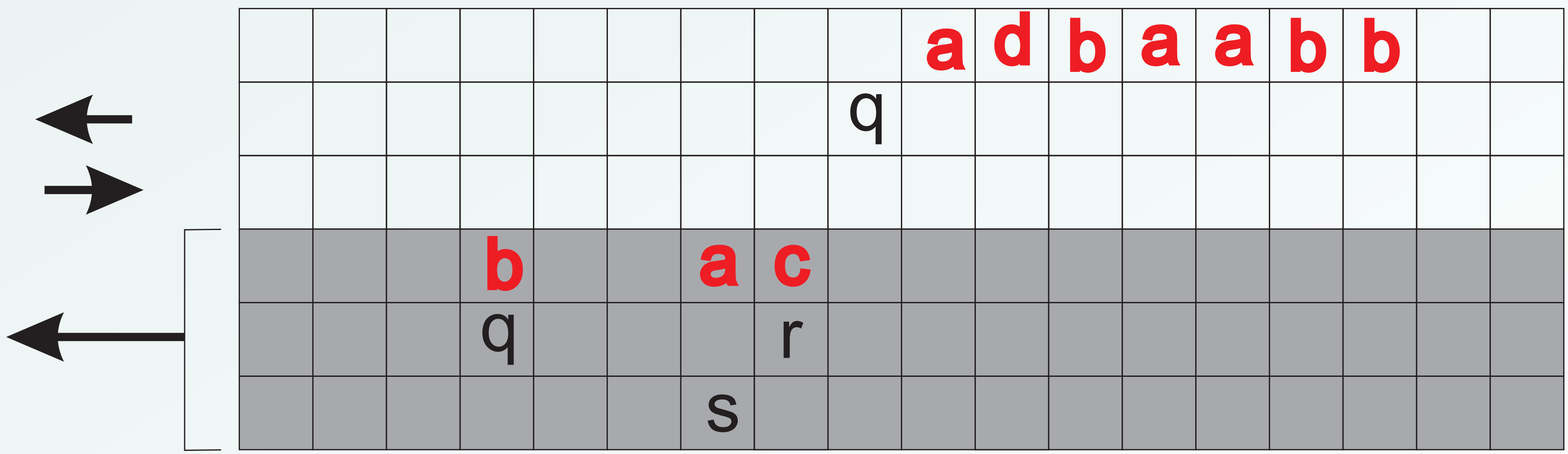


TM(r,c) = (q,a, $\leftarrow$)

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.

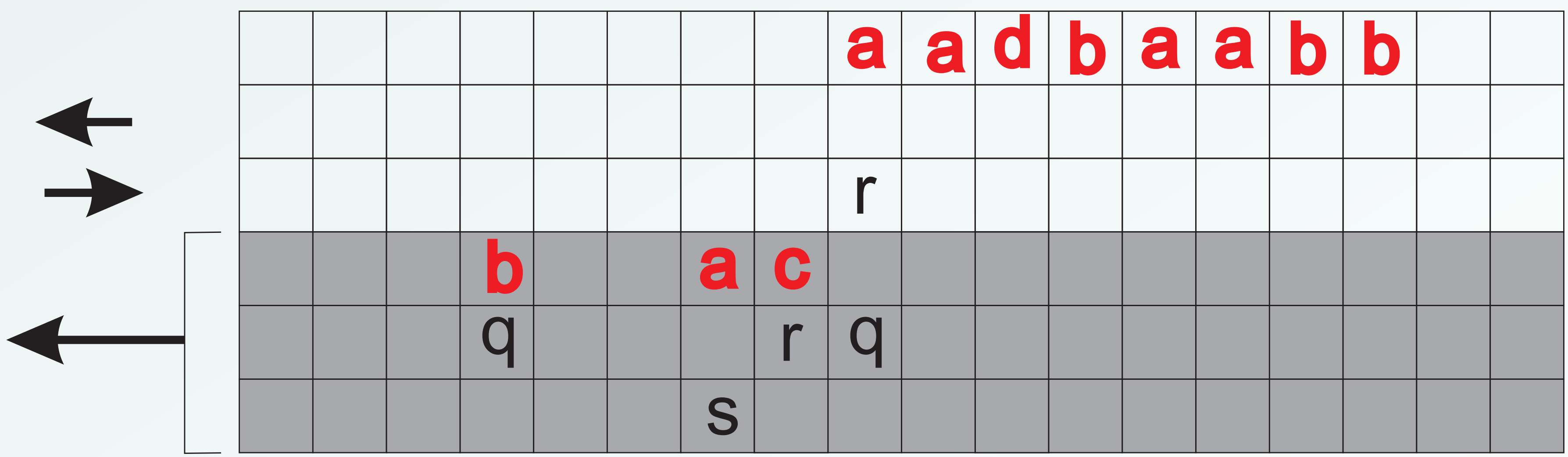


TM(q,) = (r,a,→)

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.

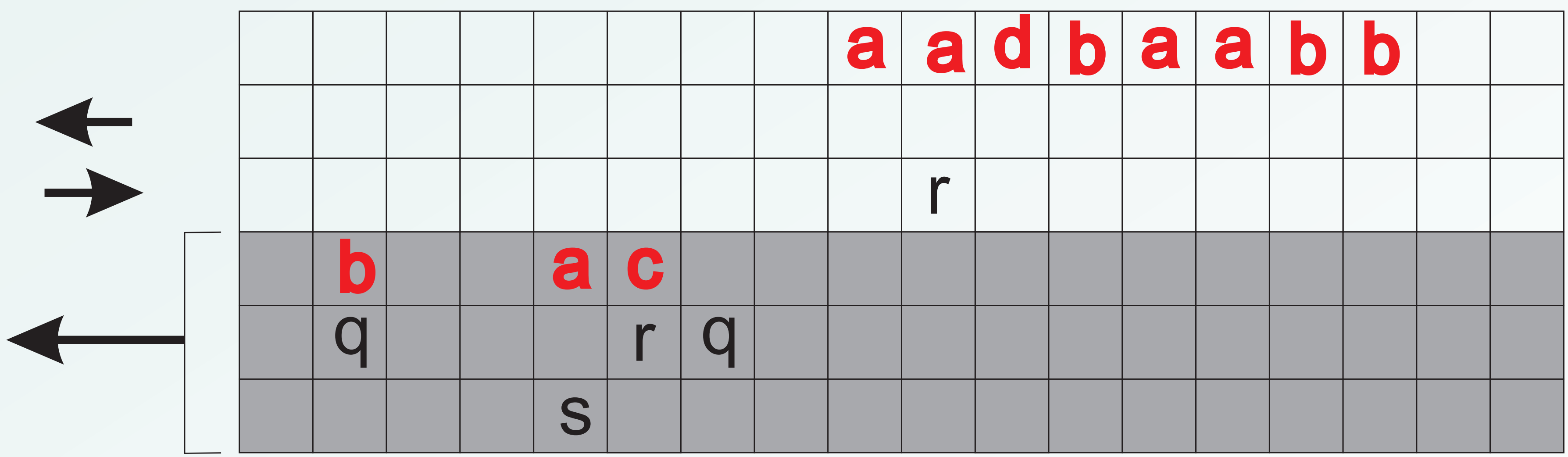


TM(q,) = (r,a,→)

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.



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

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.

Note that the missing elements correspond to situations that never occur during valid simulations of the Turing machine (for example, when the incoming new "garbage bin" is not empty).

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

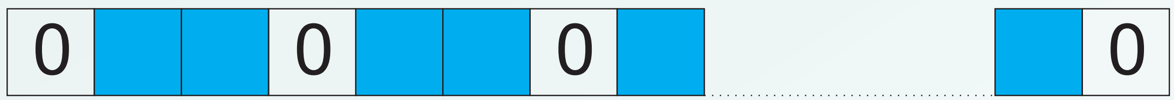
All surjective CA have **balanced** local rules: for every $a \in S$

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

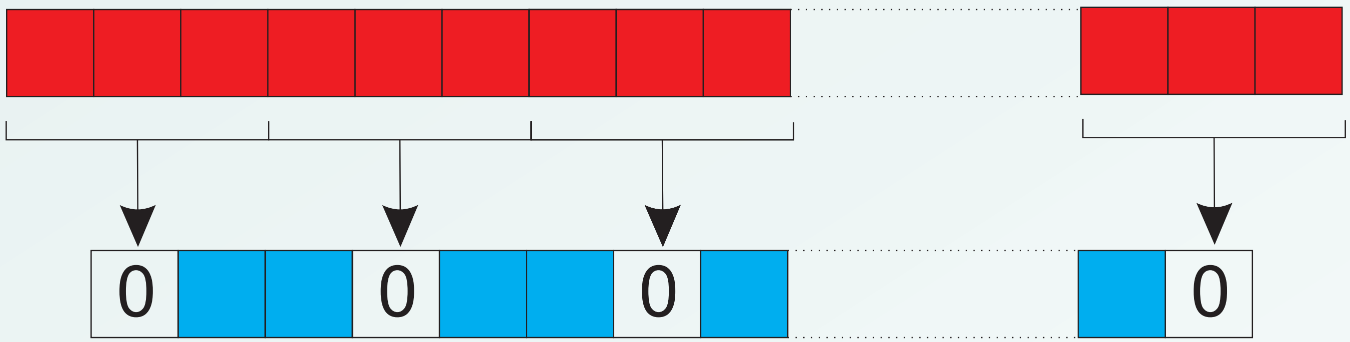
Indeed, consider a non-balanced local rule such as rule 110 where five contexts give new state 1 while only three contexts give state 0:

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

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

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

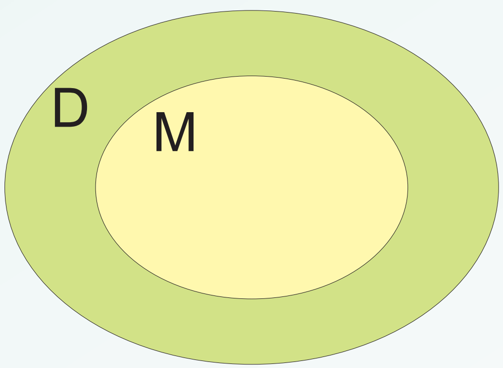
is clearly balanced, but 01001 is an orphan.

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.

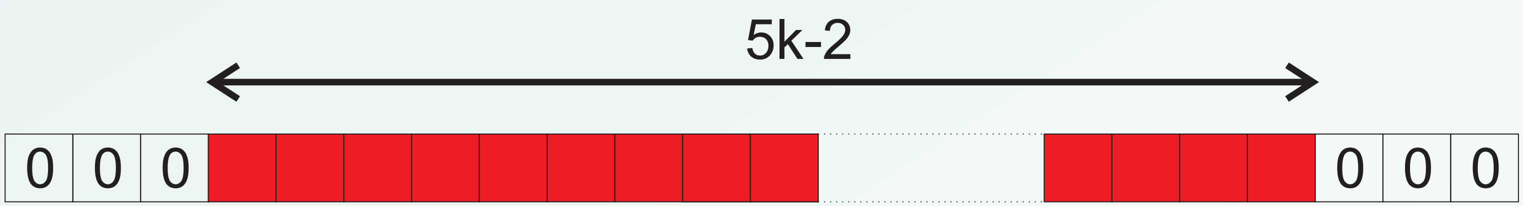
The **Garden-Of-Eden -theorem** by Moore (1962) and Myhill (1963) connects surjectivity with pre-injectivity.

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

The proof idea can be easily explained using rule 110 as a running example.

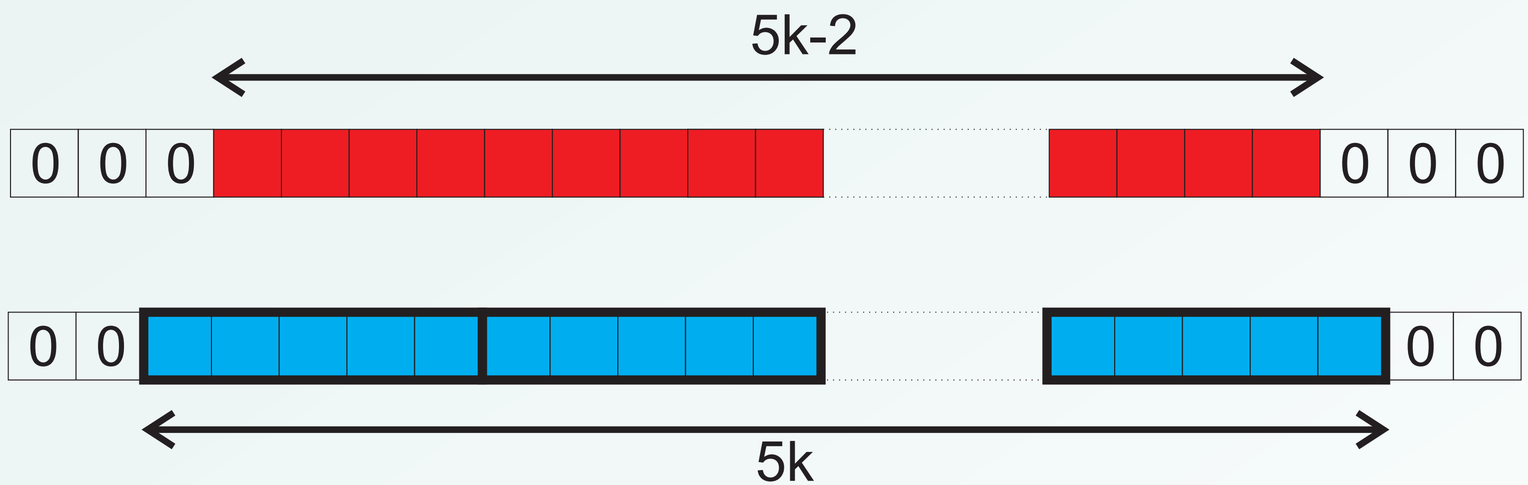
1) G not surjective $\implies$ G not pre-injective:

Since rule 110 is not surjective it has an orphan 01010 of length five. Consider a segment of length $5k-2$, for some k, and configurations c that are in state 0 outside this segment. There are $2^{5k-2} = 32^k/4$ such configurations.



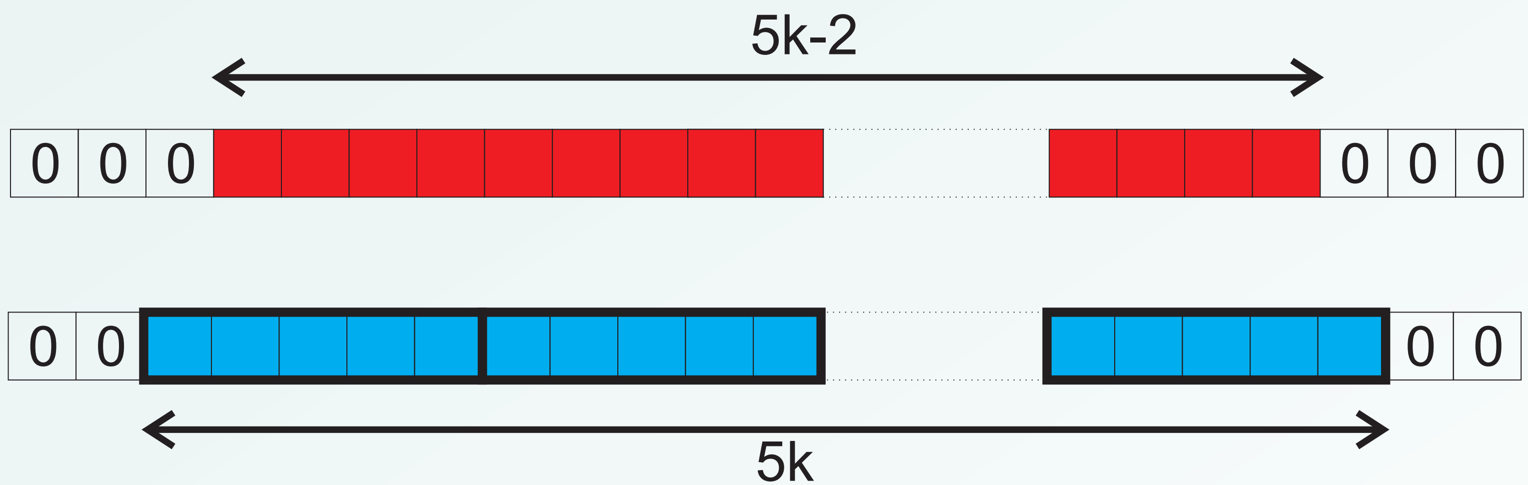
1) G not surjective $\Longrightarrow$ G not pre-injective:

The non-0 part of $G(c)$ is within a segment of length $5k$. Partition this segment into k parts of length 5. Pattern 01010 cannot appear in any part, so only $2^5 - 1 = 31$ different patterns show up in the subsegments. There are at most 31^k possible configurations $G(c)$.



1) G not surjective $\implies$ G not pre-injective:

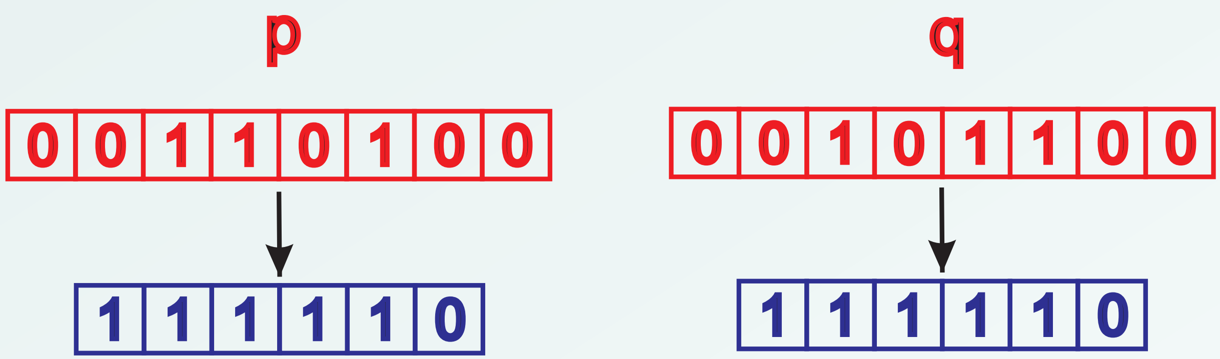
The non-0 part of $G(c)$ is within a segment of length $5k$. Partition this segment into k parts of length 5. Pattern 01010 cannot appear in any part, so only $2^5 - 1 = 31$ different patterns show up in the subsegments. There are at most 31^k possible configurations $G(c)$.



As $32^k/4 > 31^k$ for large k, there are more choices for red than blue segments. So there must exist two different red configurations with the same image. $\square$

2) G not pre-injective $\Longrightarrow$ G not surjective:

In rule 110



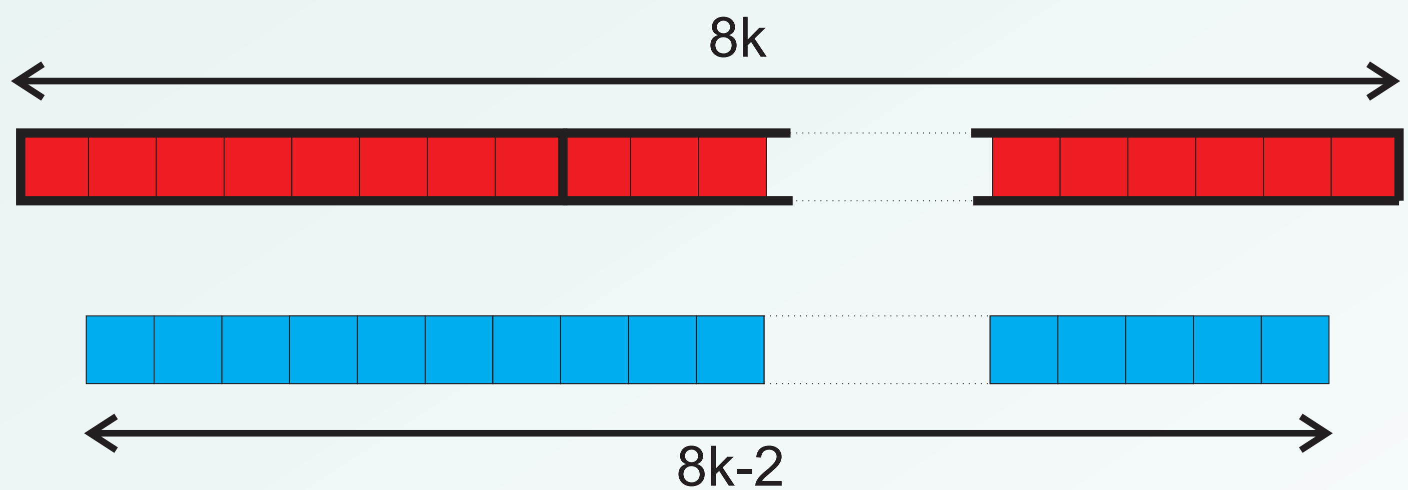
so patterns p and q of length 8 can be exchanged to each other in any configuration without affecting its image. There exist just

$$2^8 - 1 = 255$$

essentially different blocks of length 8.

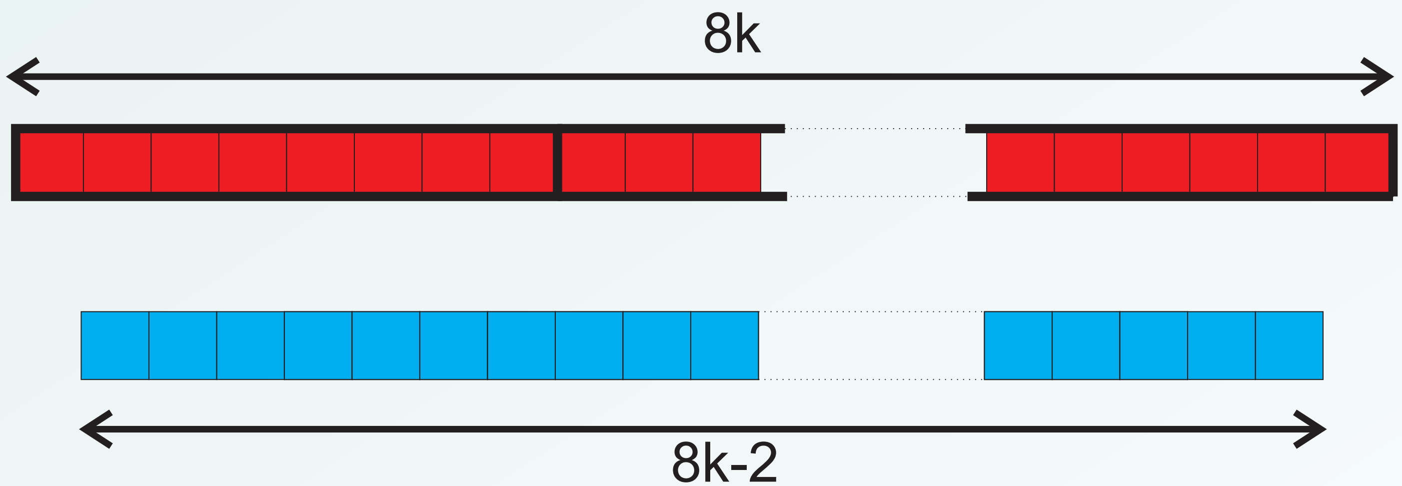
2) G not pre-injective $\Longrightarrow$ G not surjective:

Consider a segment of $8k$ cells, consisting of k parts of length 8. Patterns p and q are exchangeable, so the segment has at most 255^k different images.



2) G not pre-injective $\Longrightarrow$ G not surjective:

Consider a segment of $8k$ cells, consisting of k parts of length 8. Patterns p and q are exchangeable, so the segment has at most 255^k different images.



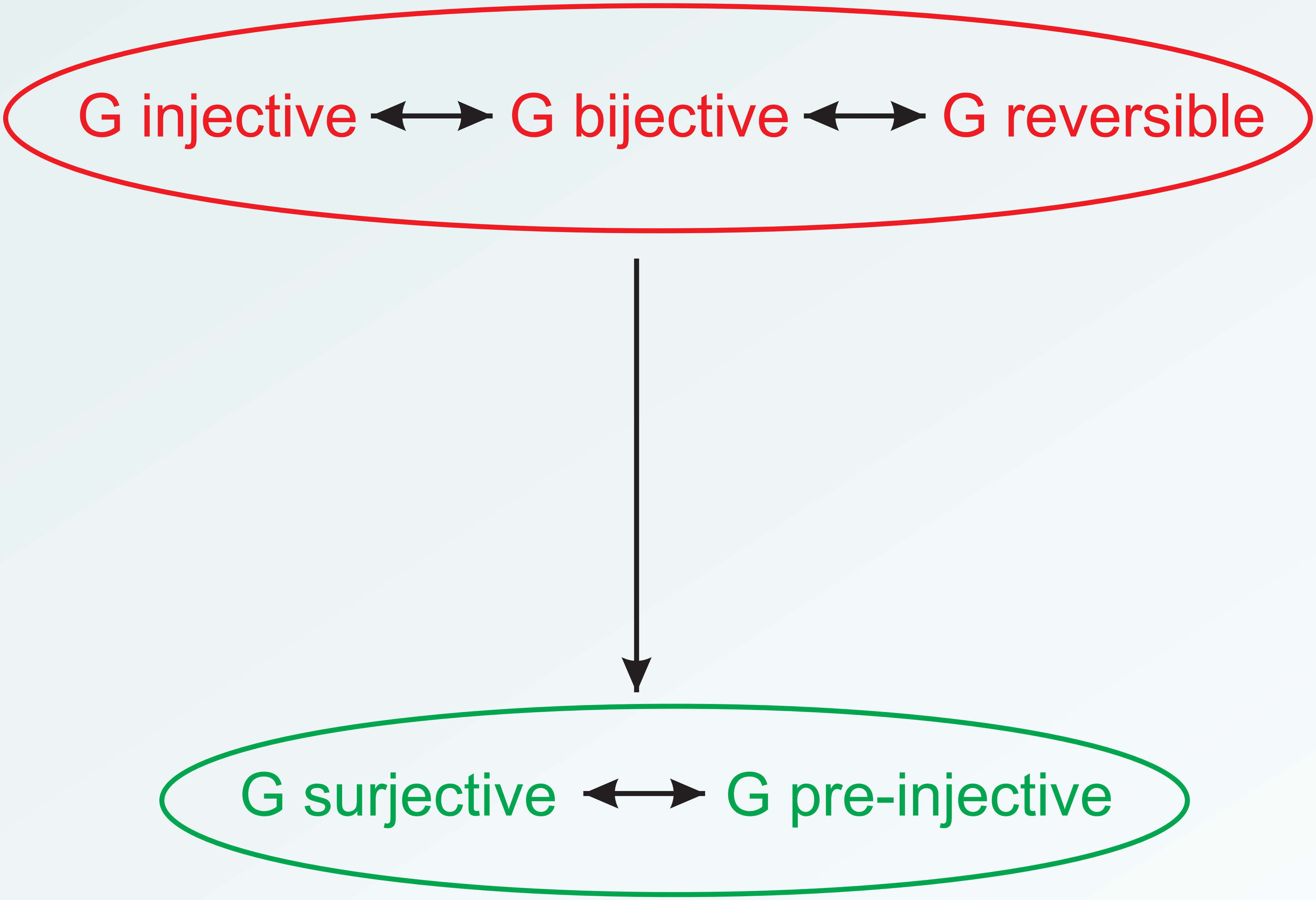
There are, however, $2^{8k-2} = 256^k/4$ different patterns of size $8k-2$. Because $255^k < 256^k/4$ for large k, there are blue patterns without any pre-image. $\square$

Garden-Of-Eden -theorem: CA G is surjective if and only if it is pre-injective.

Garden-Of-Eden -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: finite configurations

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

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

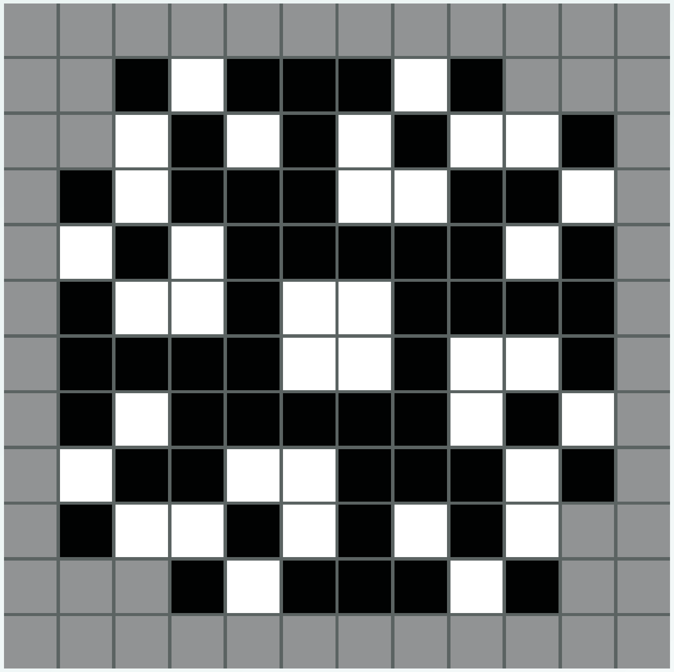
$$01001$$

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 92 cells (56 life, 36 dead):



M. Heule, C. Hartman, K. Kwekkeboom, A. Noels (2011)

Examples:

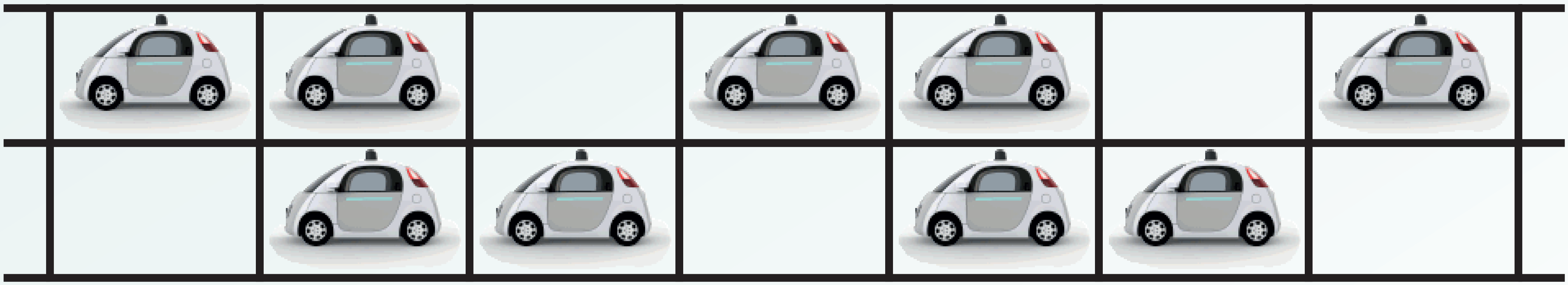
The **Traffic CA** is the elementary CA number 226.

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

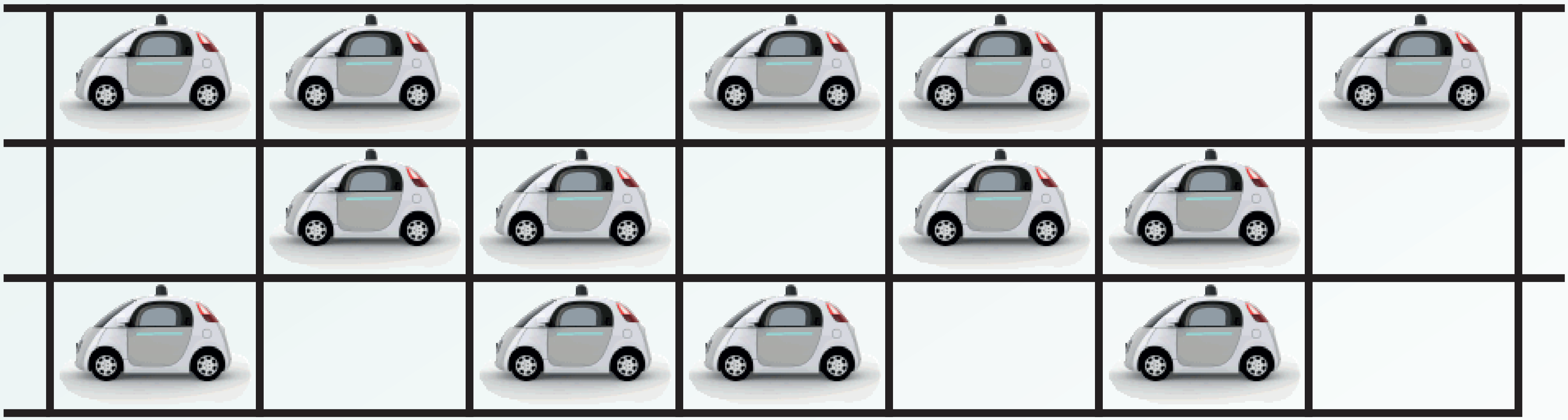
The local rule replaces pattern 01 by pattern 10.

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

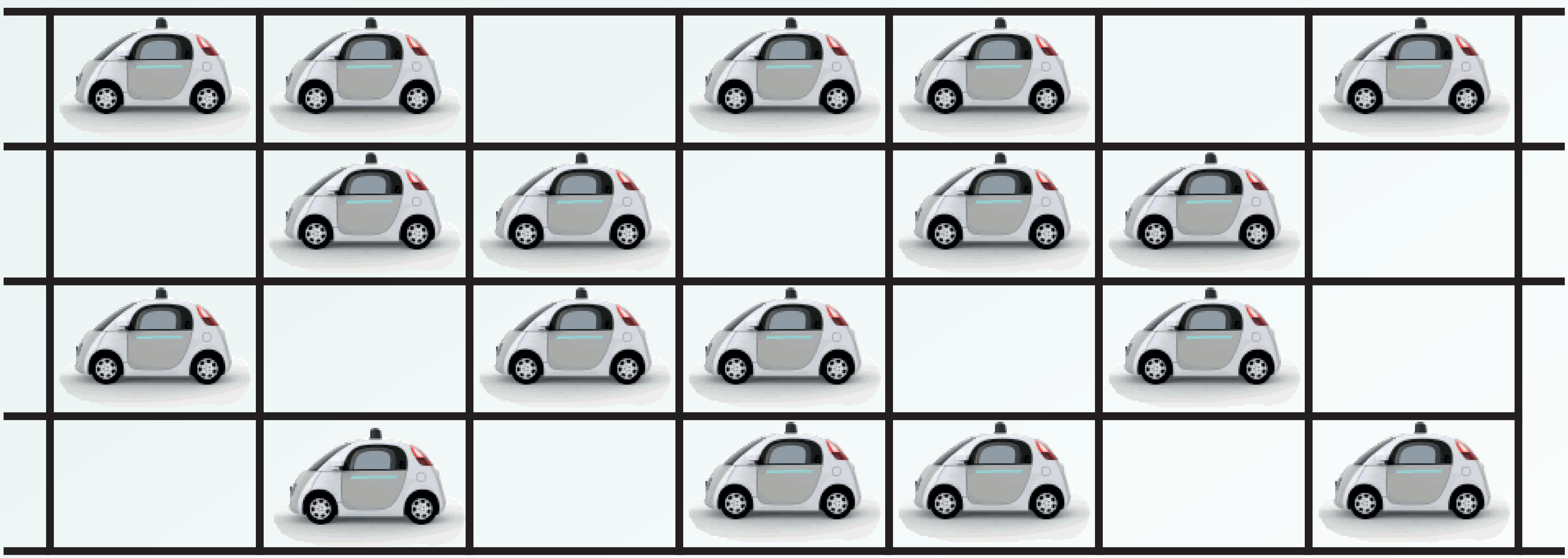
$$
\begin{array}{ccc}
111 & \longrightarrow & 1 \\
110 & \longrightarrow & 1 \\
101 & \longrightarrow & 1 \\
100 & \longrightarrow & 0 \\
011 & \longrightarrow & 0 \\
010 & \longrightarrow & 0 \\
001 & \longrightarrow & 1 \\
000 & \longrightarrow & 0
\end{array}
$$



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



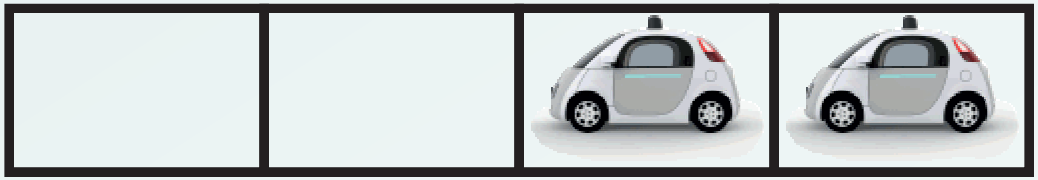
$$
\begin{array}{ccc}
111 & \longrightarrow & 1 \\
110 & \longrightarrow & 1 \\
101 & \longrightarrow & 1 \\
100 & \longrightarrow & 0 \\
011 & \longrightarrow & 0 \\
010 & \longrightarrow & 0 \\
001 & \longrightarrow & 1 \\
000 & \longrightarrow & 0
\end{array}
$$

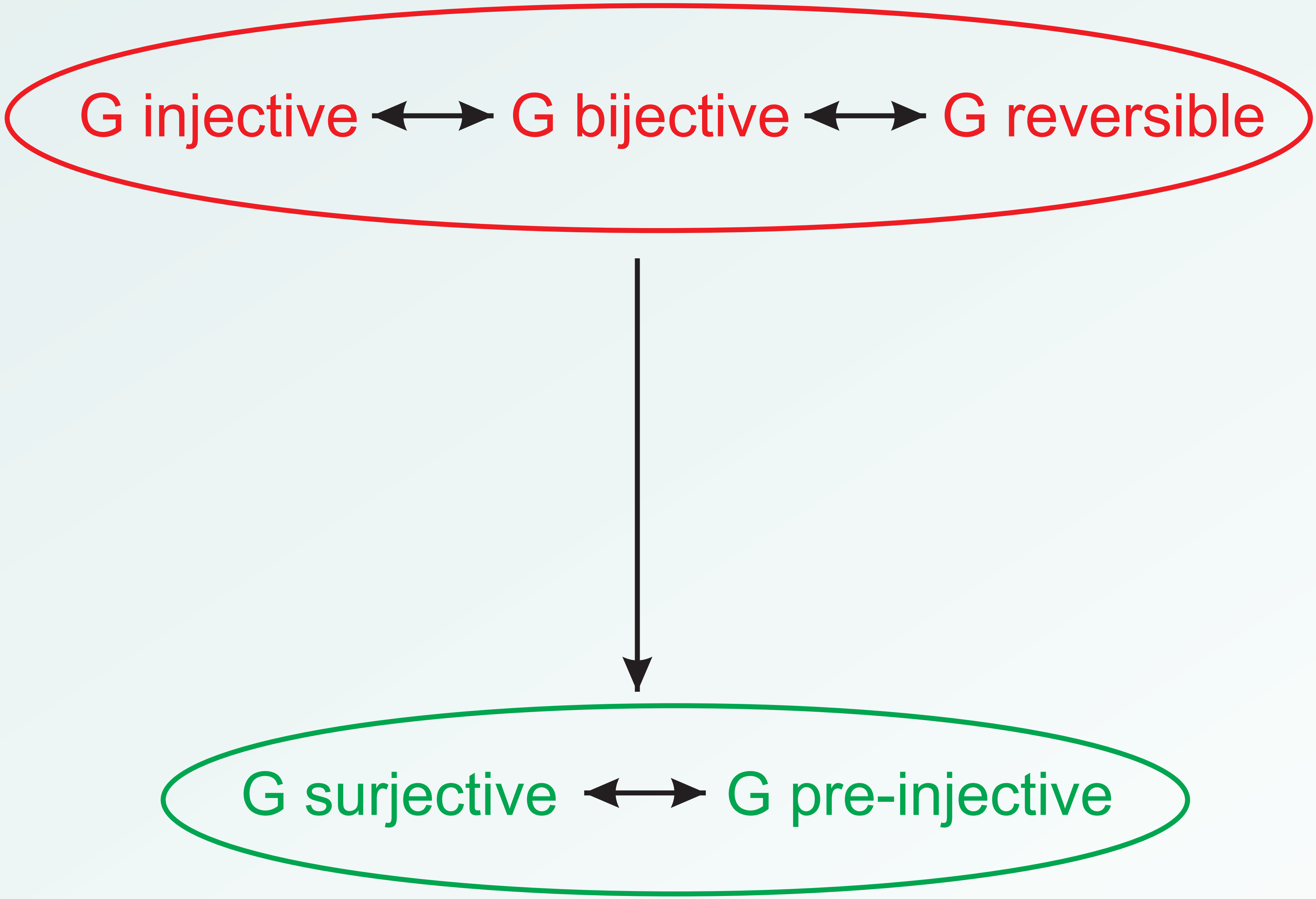


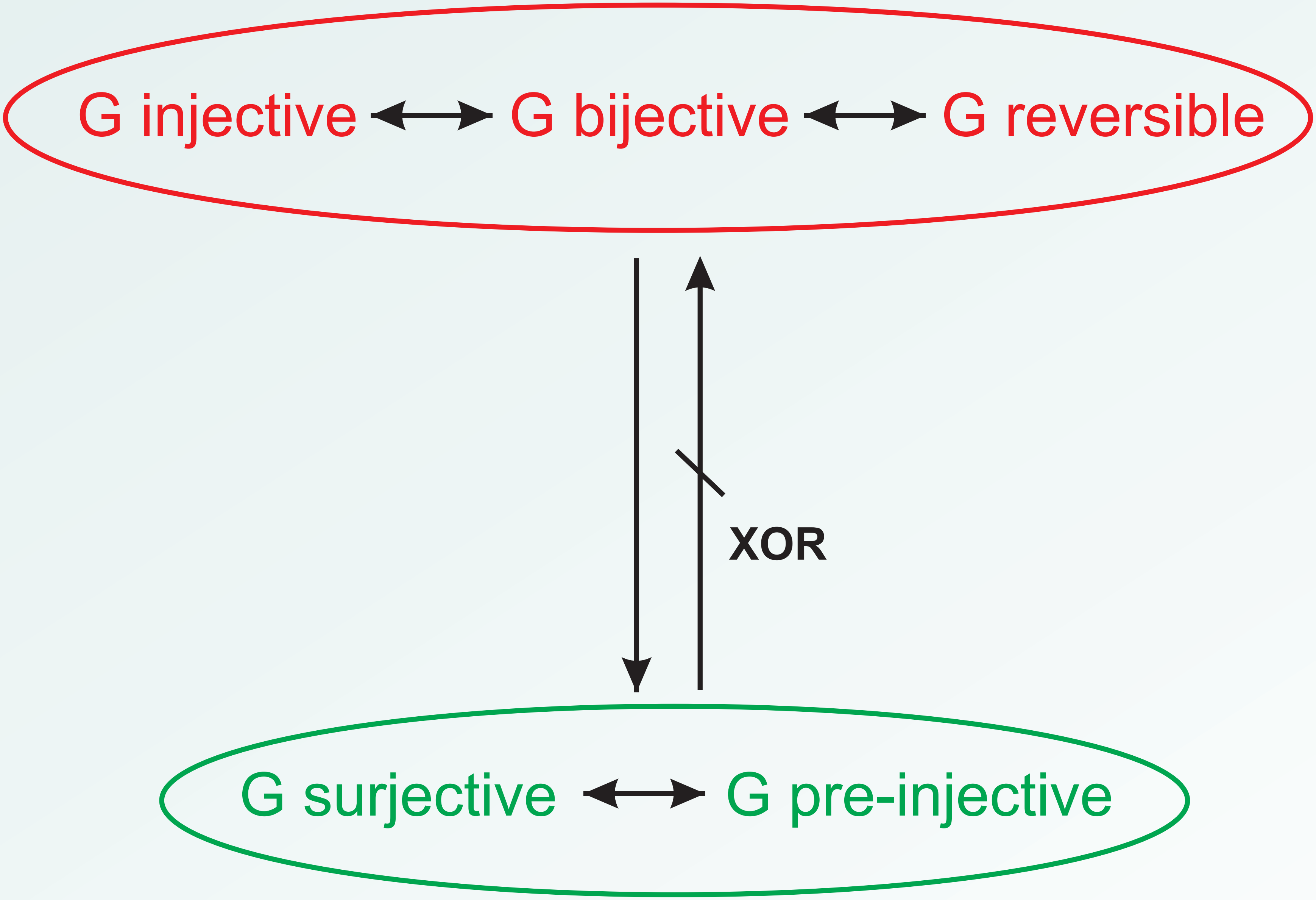
The local rule is balanced. However, there are two finite configurations with the same successor:

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

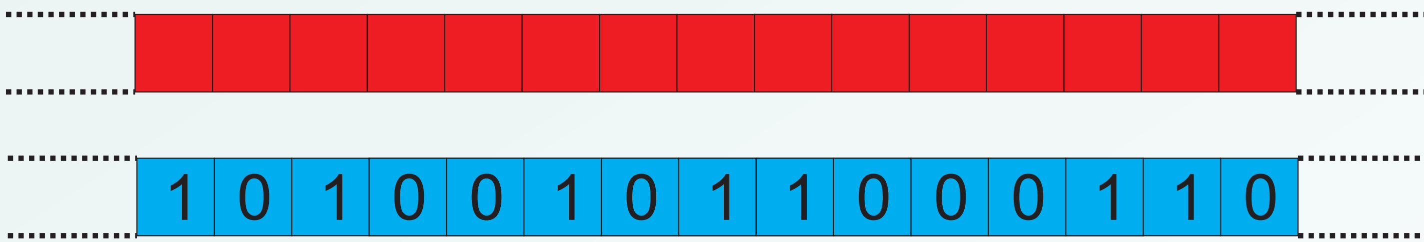
The xor-CA is the binary state CA with neighborhood $(0, 1)$ and local rule

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

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:

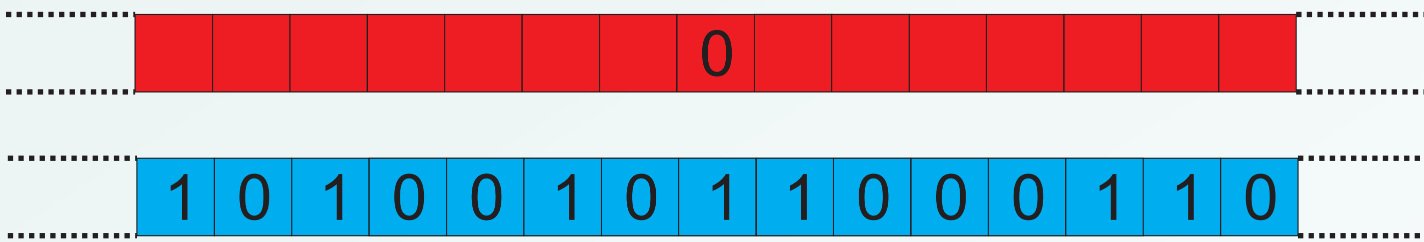


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

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:

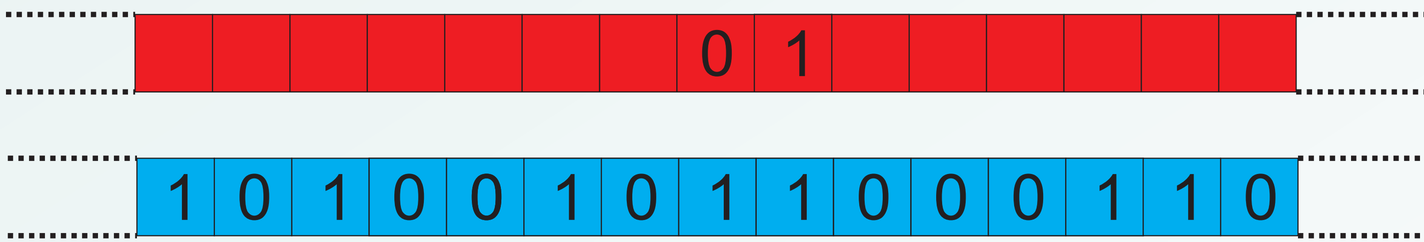


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

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:

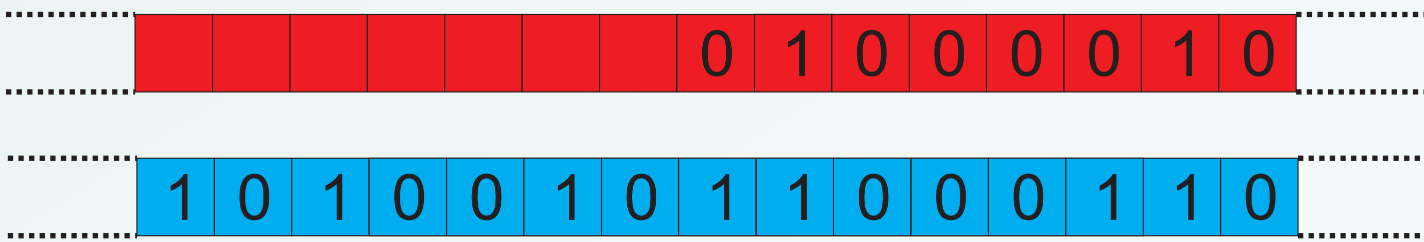


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

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:

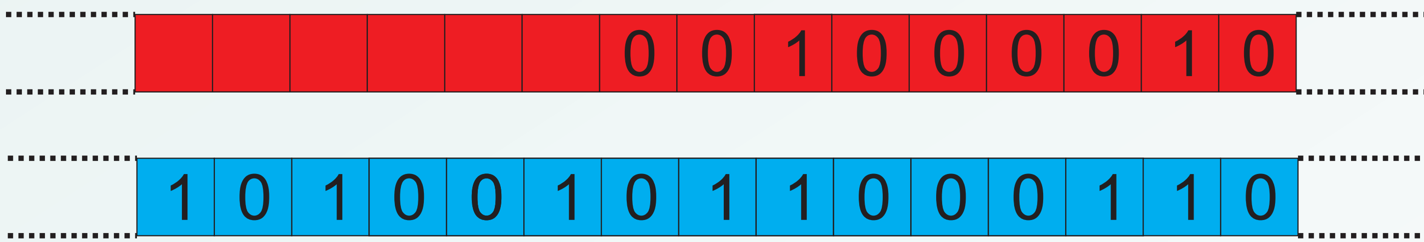


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

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:

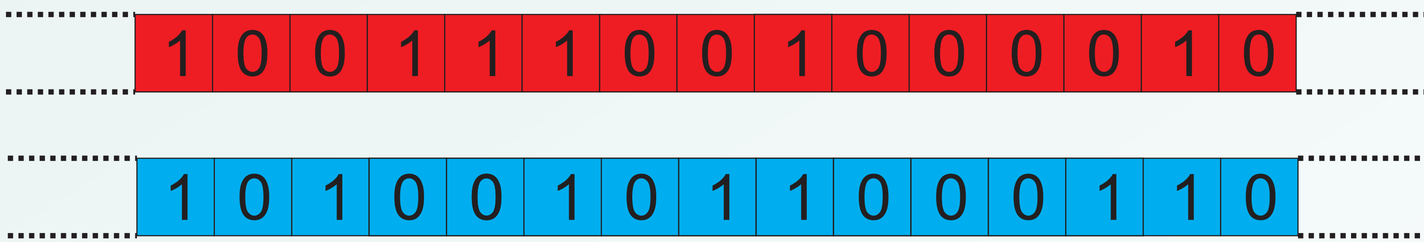


One can freely choose one value in the pre-image, after which all remaining states are uniquely determined by the **left-permutativity** and the **right-permutativity** of xor.

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.

Surjectivity and injectivity of G_P

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

Implications

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

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

We also have

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

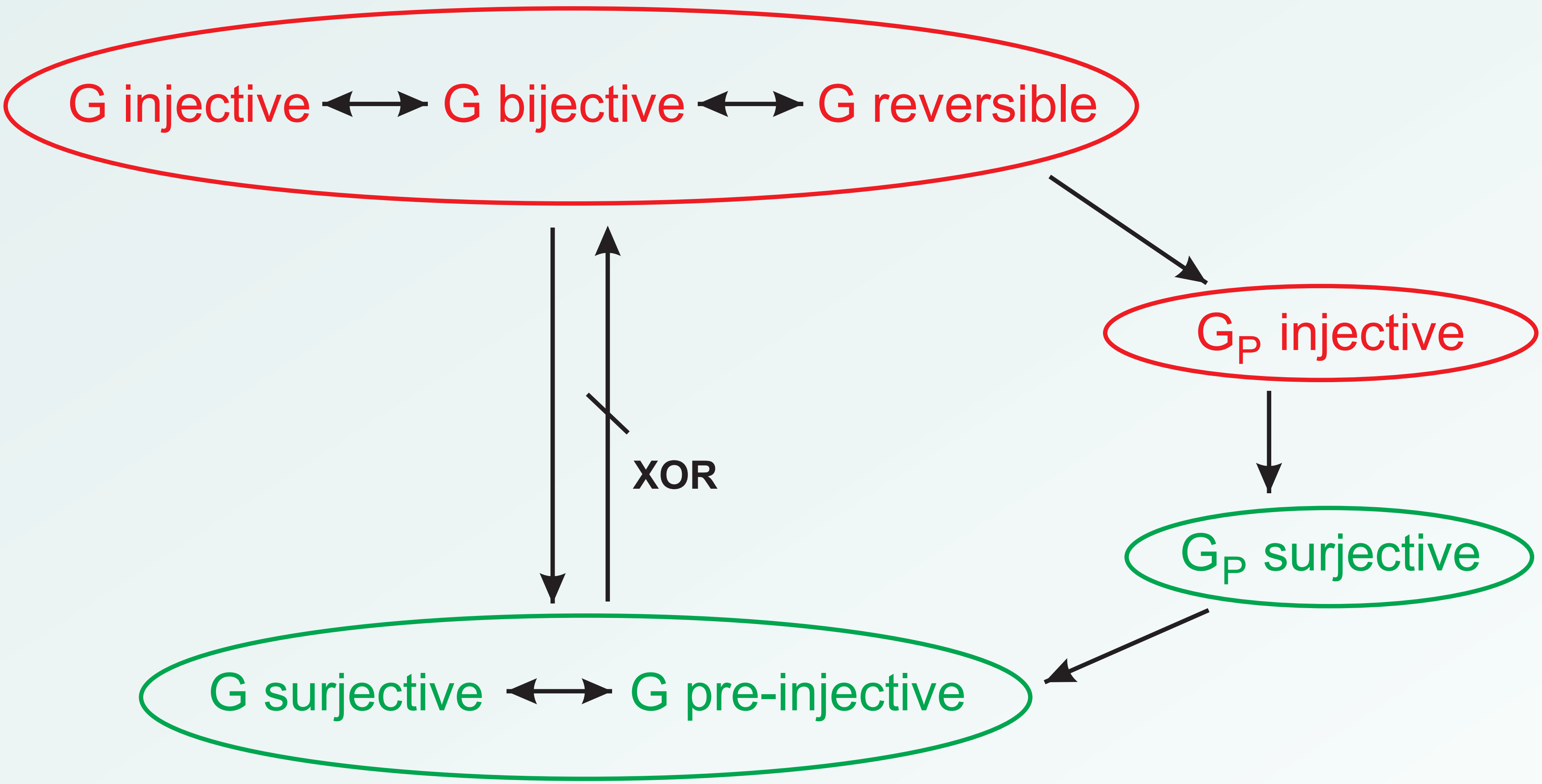
We also have

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

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:

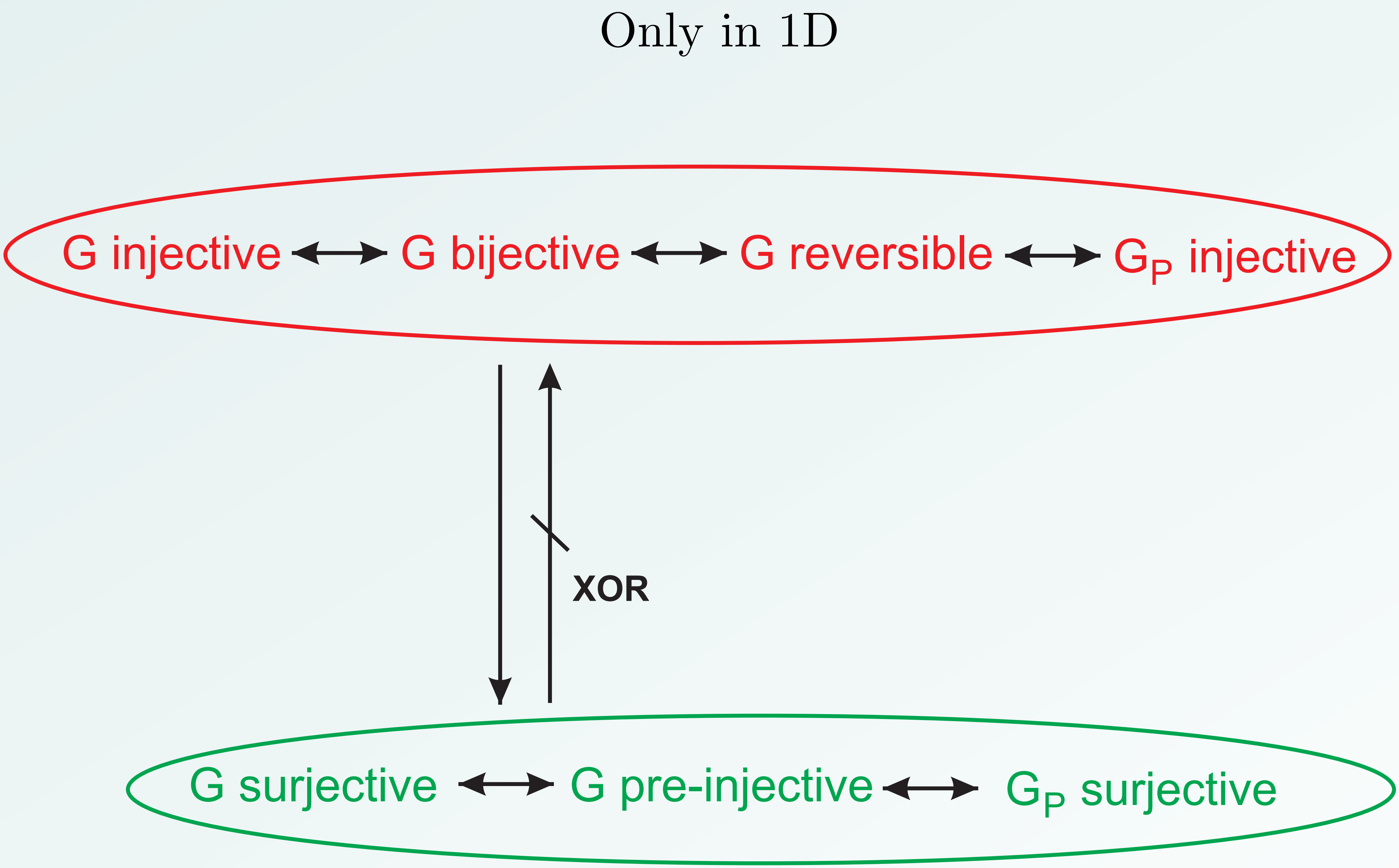
$$\mathbf{G} \text{ injective} \iff \mathbf{G_P} \text{ injective}$$

$$\mathbf{G} \text{ surjective} \iff \mathbf{G_P} \text{ surjective}$$

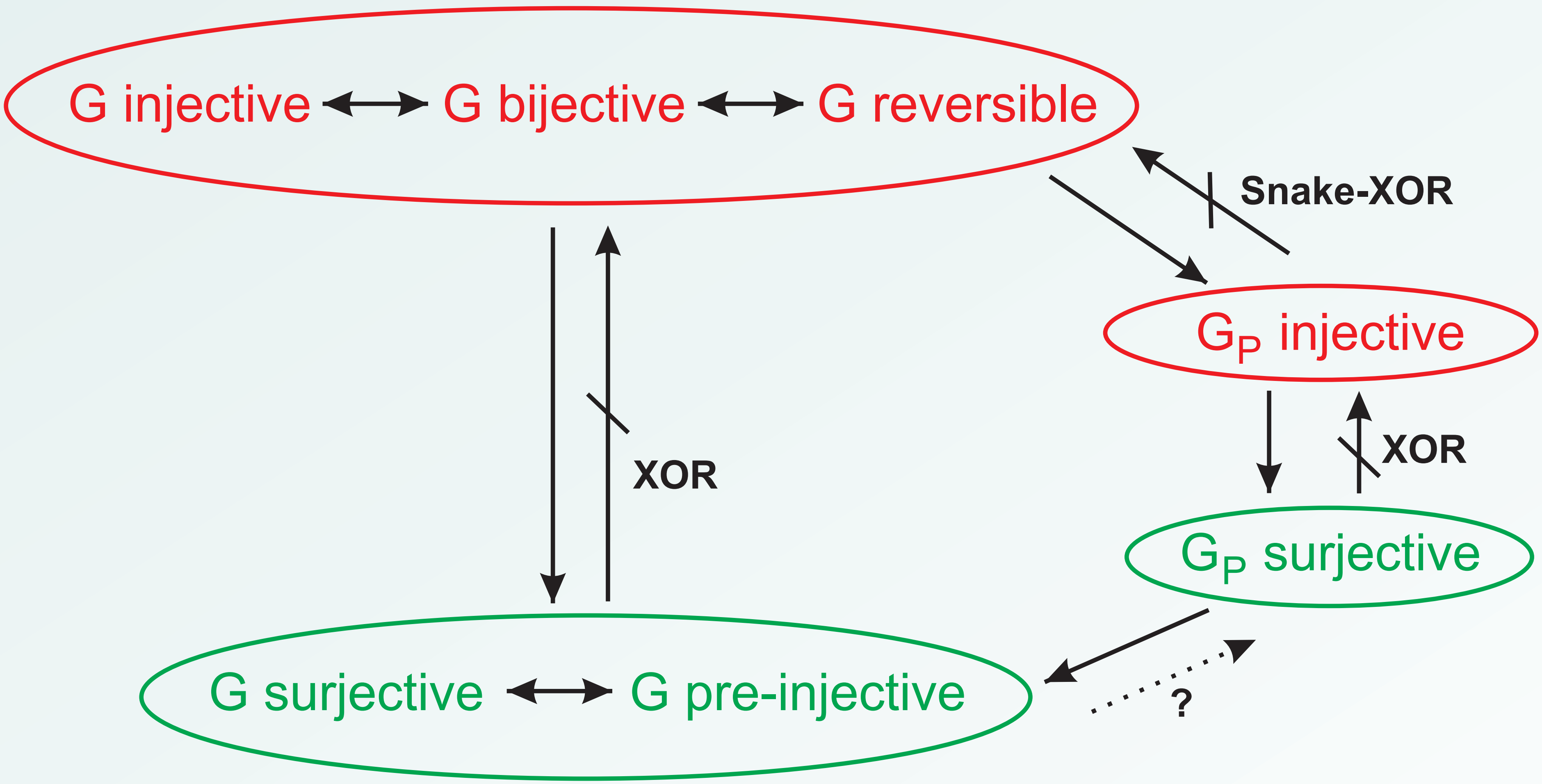
Here we get the first **dimension sensitive** property. The following equivalences are only known to hold among one-dimensional CA:

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

- The first equivalence is not true among two-dimensional CA: counter example Snake-XOR will be seen later.
- 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

We have two proofs that injective CA are surjective:

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

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

We have two proofs that injective CA are surjective:

$\mathbf{G}$ injective $\Longrightarrow$ $\mathbf{G}$ pre-injective $\Longrightarrow$ $\mathbf{G}$ surjective

$\mathbf{G}$ injective $\Longrightarrow$ $\mathbf{G_P}$ injective $\Longrightarrow$ $\mathbf{G_P}$ surjective $\Longrightarrow$ $\mathbf{G}$ surjective

It is good to have both implication chains available, if one wants to generalize results to cellular automata whose underlying grid is not $\mathbb{Z}^d$ but some other group.

- The first chain generalizes to all **amenable** groups.
- The second chain generalizes to **residually finite** groups.

A group is called **surjunctive** if every injective CA on the group is also surjective. It is not known if all groups are surjunctive.

End of the first lecture

Lecture 2: Tilings, CA and Undecidability

- Wang tiles and the undecidability of the tiling problem
- Reductions to cellular automata
- NW-determinism & one-dimensional CA
- Snakes and reversibility

Wang tiles and decidability questions

Given a cellular automaton, how to tell if it is reversible or surjective ? Is there an algorithm to decide this ? Or can we determine if the dynamics of a given CA is trivial ? Or periodic ?

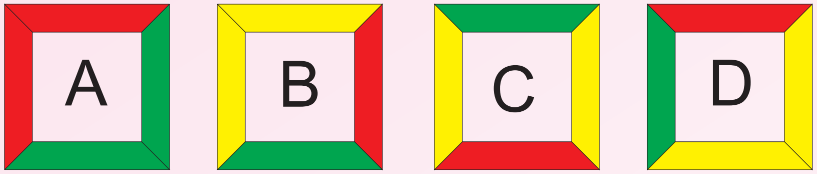
Many such algorithmic problems are **undecidable**. In some cases there is an algorithm for one-dimensional CA while the two-dimensional case is undecidable.

A useful tool: **Wang tiles** and the undecidable **tiling problem**.

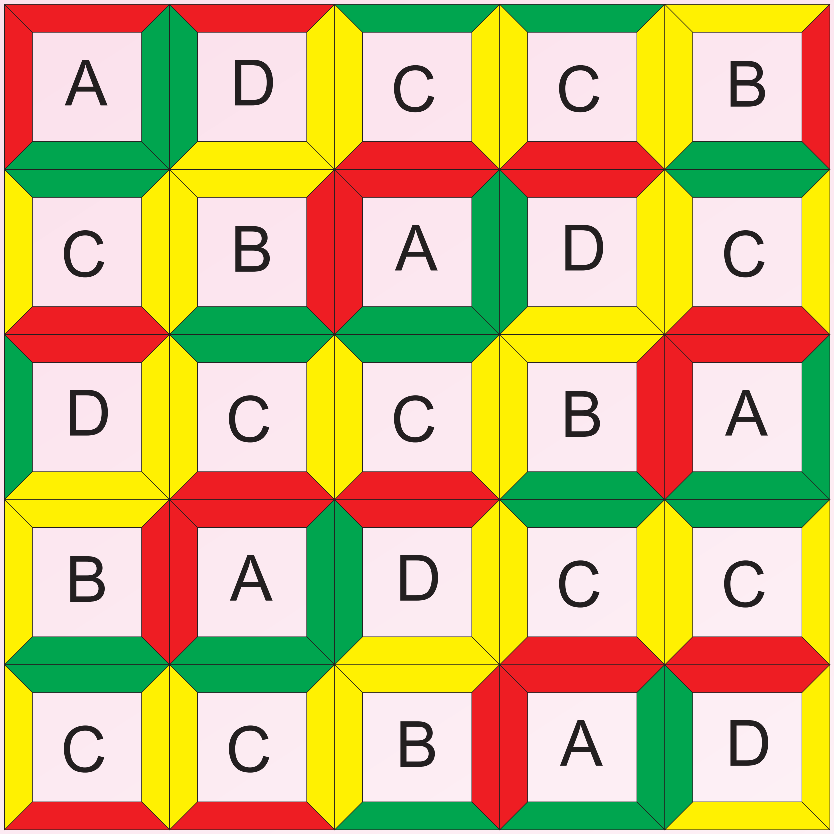
A **Wang tile** is a unit square tile with colored edges. A tile set T is a finite collection of such tiles. A valid tiling is an assignment

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

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

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

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

Follows from compactness.

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

(2) There is a **semi-algorithm** to recursively enumerate tile sets that do not admit valid tilings of the plane.

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

(2) There is a **semi-algorithm** to recursively enumerate tile sets that do not admit valid tilings of the plane.

Follows from (1): Just try tiling larger and larger squares until (if ever) a square is found that can not be tiled.

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

(2) There is a **semi-algorithm** to recursively enumerate tile sets that do not admit valid tilings of the plane.

(3) There is a **semi-algorithm** to recursively enumerate tile sets that admit a valid periodic tiling.

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

(2) There is a **semi-algorithm** to recursively enumerate tile sets that do not admit valid tilings of the plane.

(3) There is a **semi-algorithm** to recursively enumerate tile sets that admit a valid periodic tiling.

Reason: Just try tiling rectangles until (if ever) a valid tiling is found where colors on the top and the bottom match, and left and the right sides match as well.

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

(2) There is a **semi-algorithm** to recursively enumerate tile sets that do not admit valid tilings of the plane.

(3) There is a **semi-algorithm** to recursively enumerate tile sets that admit a valid periodic tiling.

Execute semi-algorithms (2) and (3) in parallel:

- If T does not tile the plane, (2) will eventually halt.
- If T admits a periodic tiling, (3) will eventually halt.

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

(2) There is a **semi-algorithm** to recursively enumerate tile sets that do not admit valid tilings of the plane.

(3) There is a **semi-algorithm** to recursively enumerate tile sets that admit a valid periodic tiling.

Execute semi-algorithms (2) and (3) in parallel:

- If T does not tile the plane, (2) will eventually halt.
- If T admits a periodic tiling, (3) will eventually halt.

Is this an algorithm that solves the **tiling problem** ?

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

(2) There is a **semi-algorithm** to recursively enumerate tile sets that do not admit valid tilings of the plane.

(3) There is a **semi-algorithm** to recursively enumerate tile sets that admit a valid periodic tiling.

Execute semi-algorithms (2) and (3) in parallel:

- If T does not tile the plane, (2) will eventually halt.
- If T admits a periodic tiling, (3) will eventually halt.

Is this an algorithm that solves the **tiling problem** ?

No! There are tile sets that fall between cases (2) and (3). They admit valid tilings but do not admit any periodic tilings.

A tile set is **aperiodic** if

- it admits valid tilings of the plane, but
- it does not admit any periodic tiling

A tile set is **aperiodic** if

- it admits valid tilings of the plane, but
- it does not admit any periodic tiling

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

A tile set is **aperiodic** if

- it admits valid tilings of the plane, but
- it does not admit any periodic tiling

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

- **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.

A tile set is **aperiodic** if

- it admits valid tilings of the plane, but
- it does not admit any periodic tiling

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

- **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 showed by computer that 11 is the smallest 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 these problems

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

Example: It is undecidable whether a given two-dimensional CA G has any fixed point configurations, that is, configurations c such that $G(c) = c$.

Example: It is undecidable whether a given two-dimensional CA G has any fixed point configurations, that is, configurations c such that $G(c) = c$.

Proof: Reduction from the tiling problem. For any given Wang tile set T (with at least two tiles) we can effectively construct a two-dimensional CA with

- state set T,
- the von Neumann -neighborhood,
- the local update rule that keeps a tile unchanged if and only if its colors match with the neighboring tiles.

Trivially, $G(c) = c$ if and only if c is a valid tiling. $\square$

Example: It is undecidable whether a given two-dimensional CA G has any fixed point configurations, that is, configurations c such that $G(c) = c$.

Proof: Reduction from the tiling problem. For any given Wang tile set T (with at least two tiles) we can effectively construct a two-dimensional CA with

- state set T,
- the von Neumann -neighborhood,
- the local update rule that keeps a tile unchanged if and only if its colors match with the neighboring tiles.

Trivially, $G(c) = c$ if and only if c is a valid tiling. $\square$

Note: For one-dimensional CA it is easily decidable whether fixed points exist.

More interesting reduction: A CA is called **nilpotent** if all configurations eventually evolve into the quiescent (=all states in state q) configuration.

Theorem (Culik, Pachl, Yu, 1989): It is undecidable whether a given two-dimensional CA is nilpotent.

More interesting reduction: A CA is called **nilpotent** if all configurations eventually evolve into the quiescent (=all states in state q) configuration.

Theorem (Culik, Pachl, Yu, 1989): It is undecidable whether a given two-dimensional CA is nilpotent.

Proof: For any given set T of Wang tiles we construct a two-dimensional CA that is nilpotent if and only if T does not admit a tiling.

For tile set T we make the following CA:

- **State set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,

For tile set T we make the following CA:

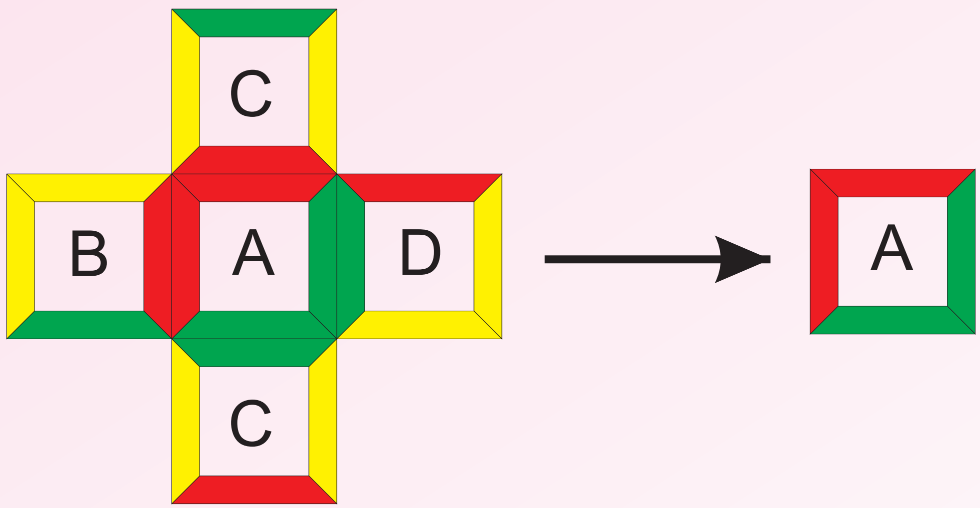
- **State set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- Von Neumann **neighborhood**,

For tile set T we make the following CA:

- **State set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- Von Neumann **neighborhood**,
- The **local rule** keeps state unchanged if all states in the neighborhood are tiles and the tiling constraint is satisfied. In all other cases the new state is q.

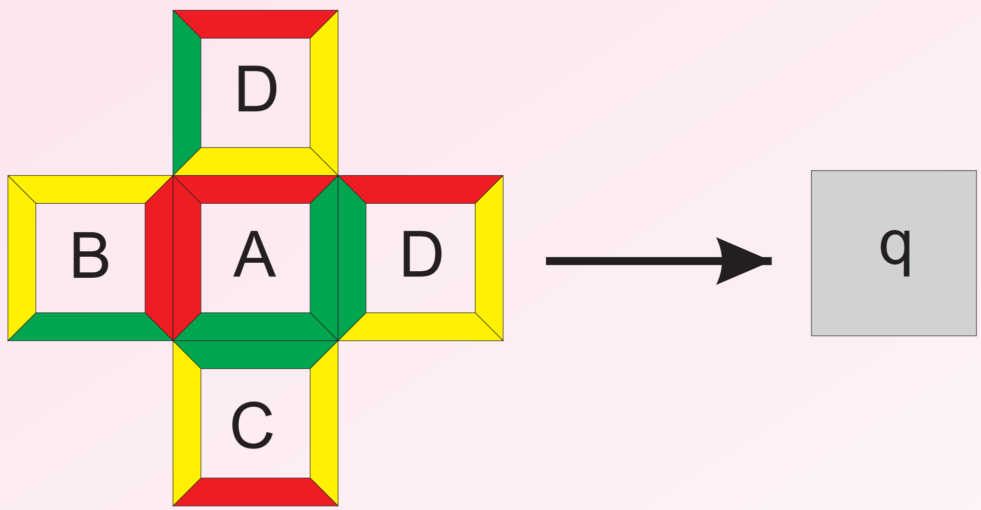
For tile set T we make the following CA:

- **State set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- Von Neumann **neighborhood**,
- The **local rule** keeps state unchanged if all states in the neighborhood are tiles and the tiling constraint is satisfied. In all other cases the new state is q.



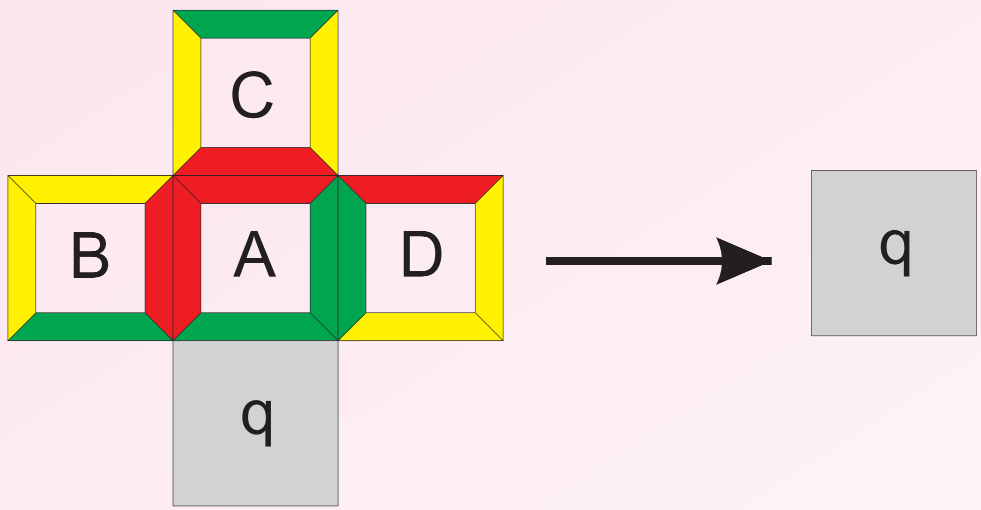
For tile set T we make the following CA:

- **State set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- Von Neumann **neighborhood**,
- The **local rule** keeps state unchanged if all states in the neighborhood are tiles and the tiling constraint is satisfied. In all other cases the new state is q.



For tile set T we make the following CA:

- **State set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- Von Neumann **neighborhood**,
- The **local rule** keeps state unchanged if all states in the neighborhood are tiles and the tiling constraint is satisfied. In all other cases the new state is q.



For tile set T we make the following CA:

- **State set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- Von Neumann **neighborhood**,
- The **local rule** keeps state unchanged if all states in the neighborhood are tiles and the tiling constraint is satisfied. In all other cases the new state is q.

For tile set T we make the following CA:

- **State set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- Von Neumann **neighborhood**,
- The **local rule** keeps state unchanged if all states in the neighborhood are tiles and the tiling constraint is satisfied. In all other cases the new state is q.

$\Longrightarrow$ If T admits a tiling c then c is a non-quiescent fixed point of the CA. So the CA is not nilpotent.

For tile set T we make the following CA:

- **State set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- Von Neumann **neighborhood**,
- The **local rule** keeps state unchanged if all states in the neighborhood are tiles and the tiling constraint is satisfied. In all other cases the new state is q.

$\Longrightarrow$ If T admits a tiling c then c is a non-quiescent fixed point of the CA. So the CA is not nilpotent.

$\Longleftarrow$ If T does not admit a valid tiling then every $n \times n$ square contains a tiling error, for some n. State q propagates, so in at most $2n$ steps all cells are in state q. The CA is nilpotent. $\square$

If we do the previous construction for an aperiodic tile set T we obtain a two-dimensional CA in which

- every periodic configuration becomes eventually quiescent, but
- there are some non-periodic fixed points.

NW-deterministic tiles

Tilings relate naturally to two-dimensional CA.

What about **one-dimensional CA** ?

NW-deterministic tiles

Tilings relate naturally to two-dimensional CA.

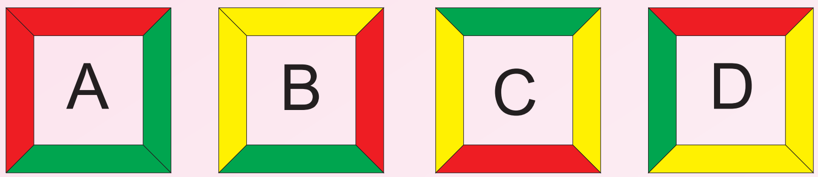
What about **one-dimensional CA** ?

We can strengthen Berger's result so that the **nilpotency** can be proved undecidable for one-dimensional CA as well.

The basic idea is to view **space-time diagrams** of one-dimensional CA as tilings.

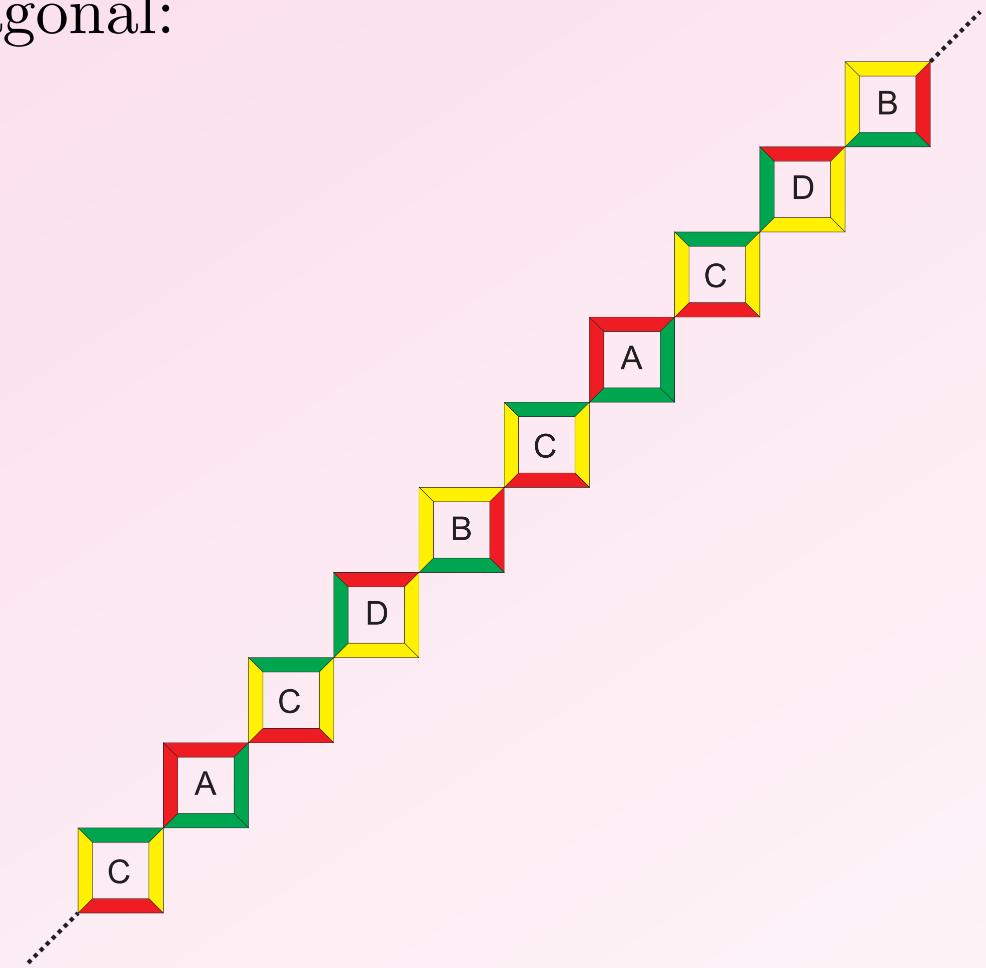
Tile set T is **NW-deterministic** if no two tiles have identical colors on their top edges and on their left edges. In a valid tiling the left and the top neighbor of a tile uniquely determine the tile.

For example, our sample tile set

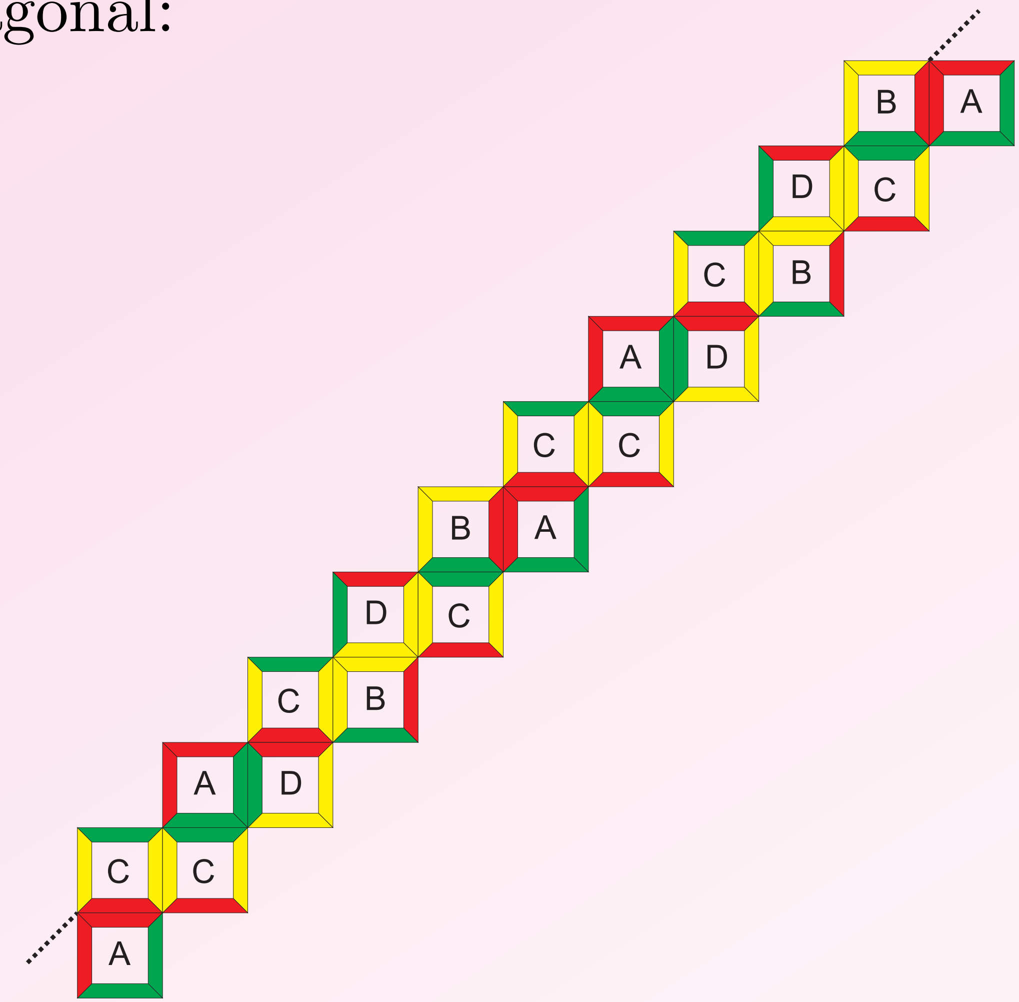


is NW-deterministic.

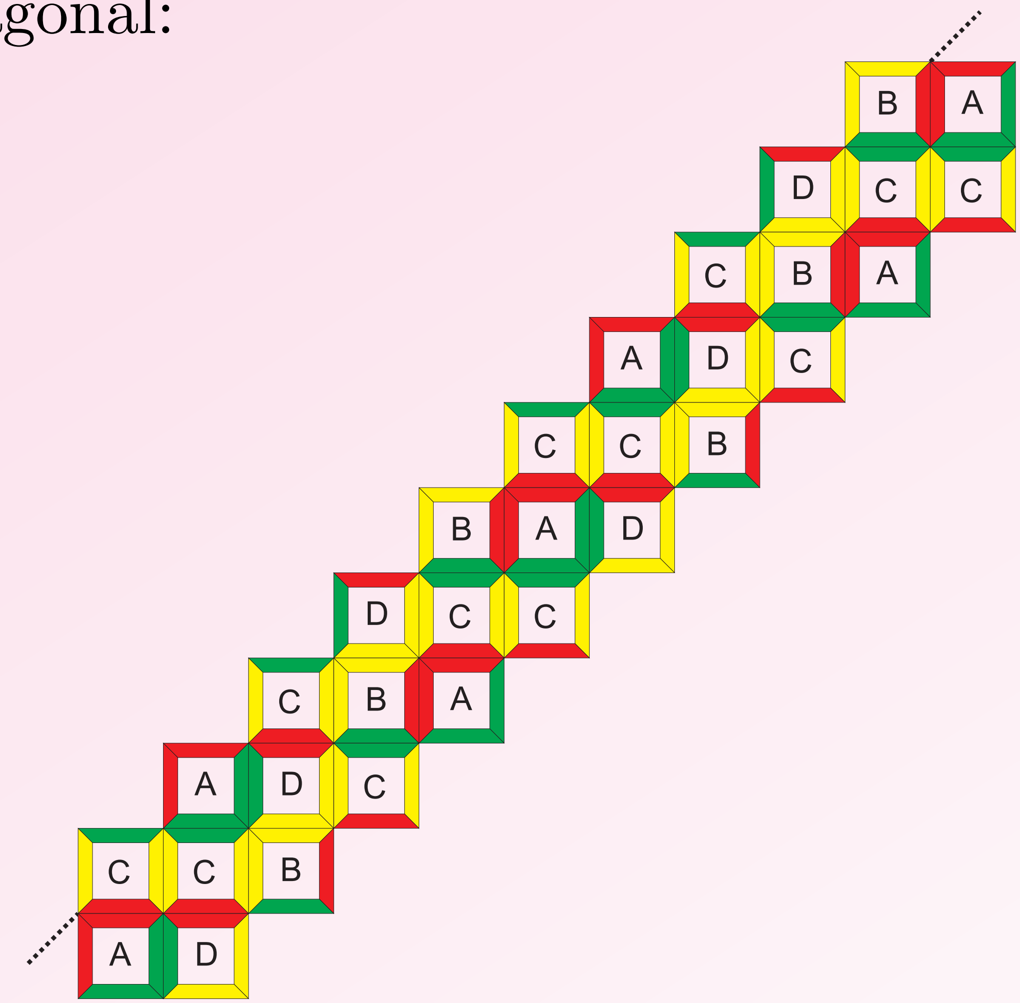
In any valid tiling by NW-deterministic tiles, NE-to-SW diagonals uniquely determine the next diagonal below them. The tiles of the next diagonal are determined locally from the previous diagonal:



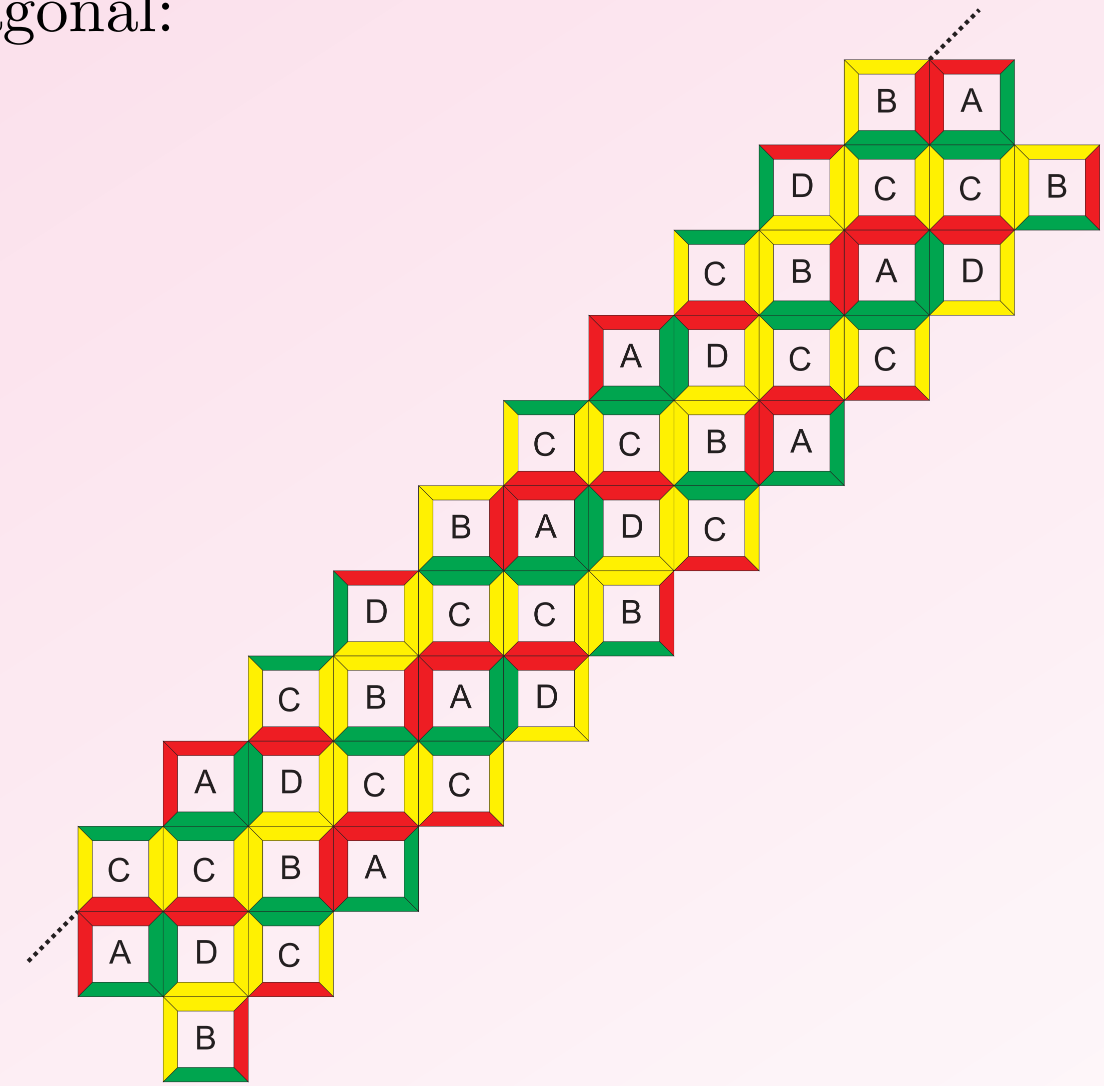
In any valid tiling by NW-deterministic tiles, NE-to-SW diagonals uniquely determine the next diagonal below them. The tiles of the next diagonal are determined locally from the previous diagonal:

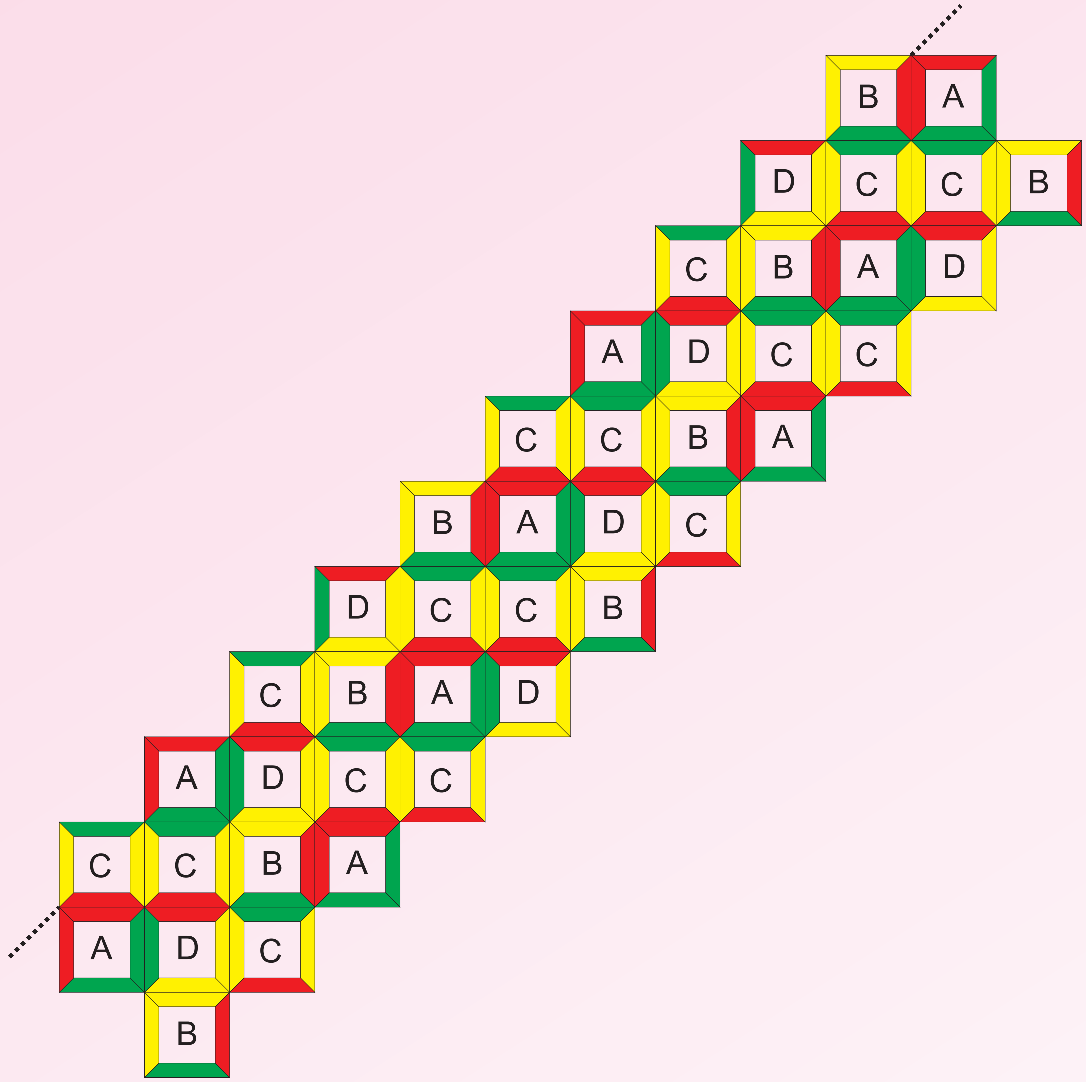


In any valid tiling by NW-deterministic tiles, NE-to-SW diagonals uniquely determine the next diagonal below them. The tiles of the next diagonal are determined locally from the previous diagonal:



In any valid tiling by NW-deterministic tiles, NE-to-SW diagonals uniquely determine the next diagonal below them. The tiles of the next diagonal are determined locally from the previous diagonal:





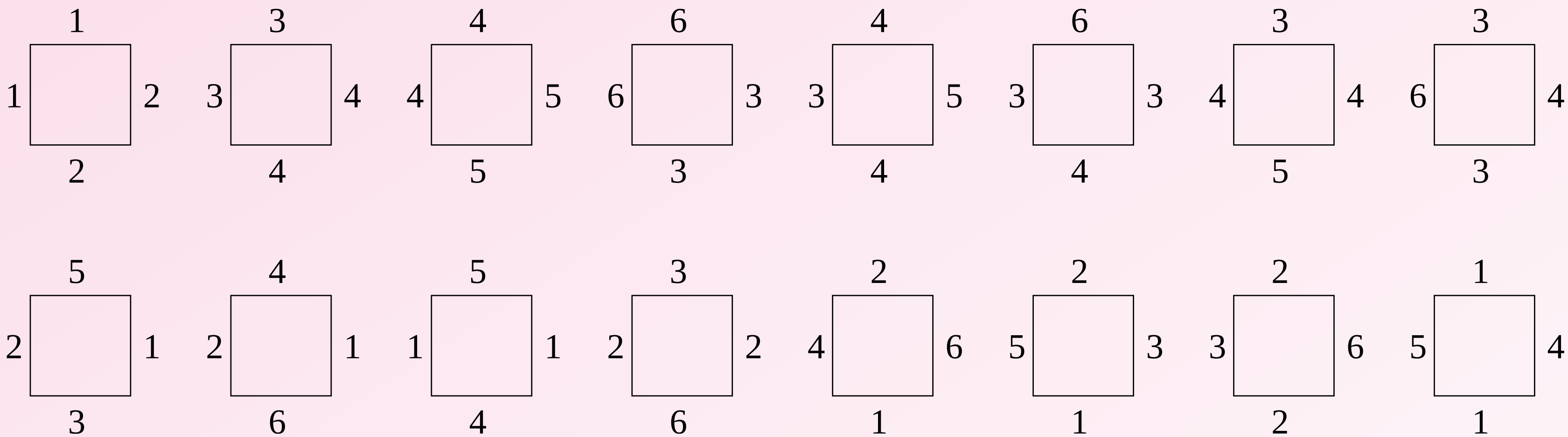
If diagonals are interpreted as configurations of a one-dimensional CA, valid tilings represent space-time diagrams.

But are there complex NW-deterministic tile sets? Are they interesting?

But are there complex NW-deterministic tile sets? Are they interesting?

YES!

1. There are **aperiodic** NW-deterministic tiles sets:

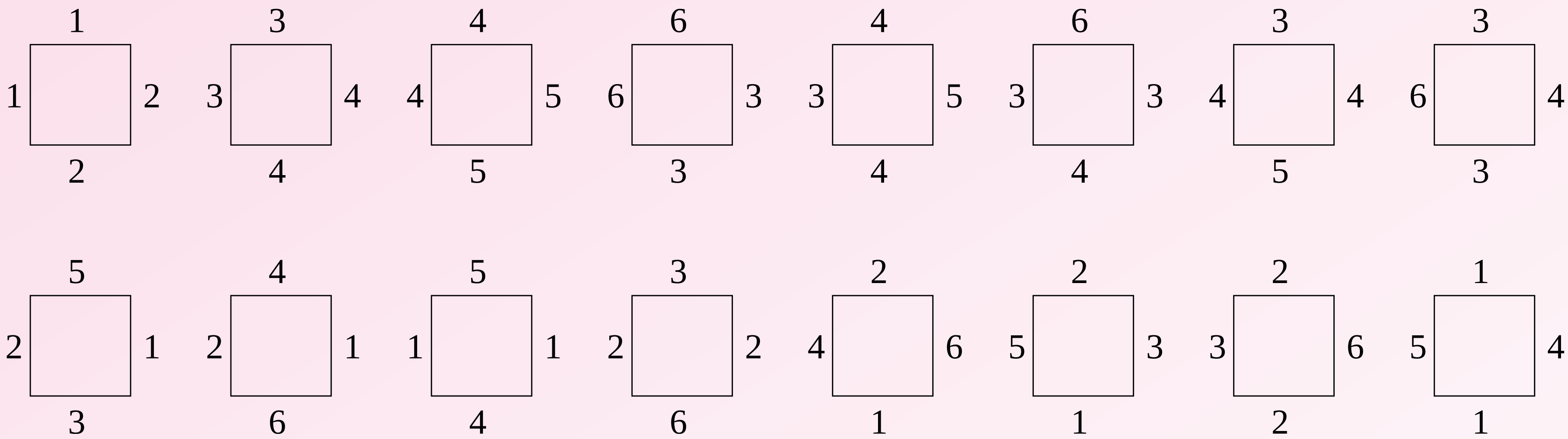


Amman's 16 tile aperiodic tile set

But are there complex NW-deterministic tile sets? Are they interesting?

YES!

1. There are **aperiodic** NW-deterministic tiles sets:



Amman's 16 tile aperiodic tile set

2. With a bit of effort (proof omitted):

Theorem: The tiling problem is undecidable among NW-deterministic tile sets.

1D nilpotency is undecidable: For any given NW-deterministic tile set T we construct a one-dimensional CA whose

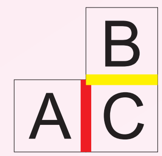
1D nilpotency is undecidable: For any given NW-deterministic tile set T we construct a one-dimensional CA whose

- **state set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,

1D nilpotency is undecidable: For any given NW-deterministic tile set T we construct a one-dimensional CA whose

- **state set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- **neighborhood** is $(0, 1)$,

1D nilpotency is undecidable: For any given NW-deterministic tile set T we construct a one-dimensional CA whose

- **state set** is $S = T \cup \{q\}$ where q is a new symbol $q \notin T$,
- **neighborhood** is $(0, 1)$,
- **local rule** $f : S^2 \longrightarrow S$ is defined as follows:
  - $f(A, B) = C$ if the colors match in 

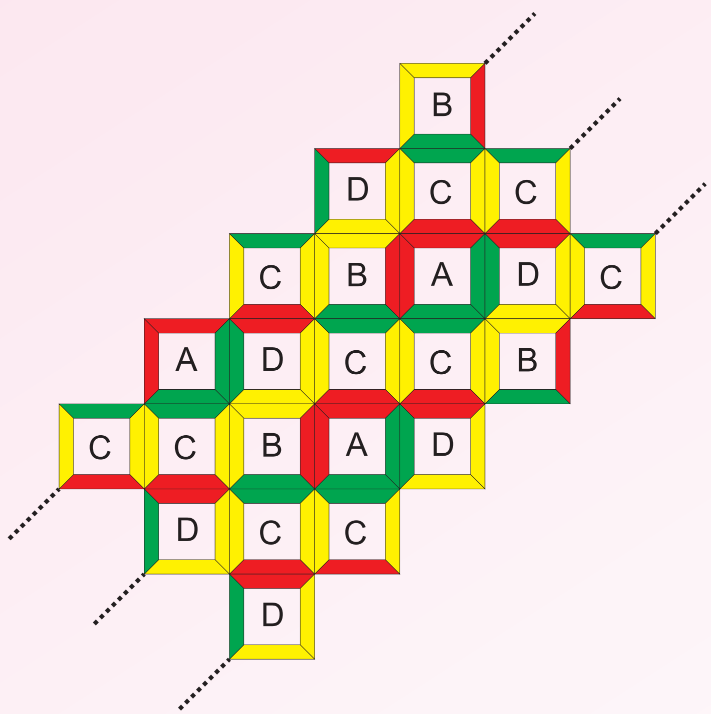
  - $f(A, B) = q$ if $A = q$ or $B = q$ or no matching tile C exists.

Claim: The CA is nilpotent if and only if T does not admit a tiling.

Claim: The CA is nilpotent if and only if T does not admit a tiling.

Proof:

$\Longrightarrow$ If T **admits a tiling** c then diagonals of c are configurations that never evolve into the quiescent configuration. So the CA is **not nilpotent**.



Claim: The CA is nilpotent if and only if T does not admit a tiling.

Proof:

$\Longrightarrow$ If T **admits a tiling** c then diagonals of c are configurations that never evolve into the quiescent configuration. So the CA is **not nilpotent**.

$\Longleftarrow$ If T **does not admit a tiling** then every $n \times n$ square contains a tiling error, for some n. Hence state q is created inside every segment of length n.

Since q spreads, the whole configuration becomes eventually quiescent. The CA is **nilpotent**.

The tiling problem is undecidable for NW-deterministic tile sets, so

Theorem: It is undecidable whether a given one-dimensional CA is nilpotent. $\square$

The tiling problem is undecidable for NW-deterministic tile sets, so

Theorem: It is undecidable whether a given one-dimensional CA is nilpotent. $\square$

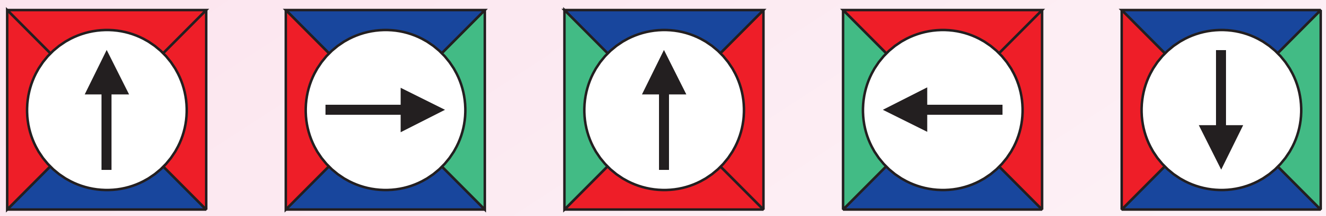
If we do the previous construction using an aperiodic set then we have an interesting one-dimensional CA:

- all periodic configurations eventually die, but
- there are non-periodic configurations that never create a quiescent state in any cell.

SNAKES

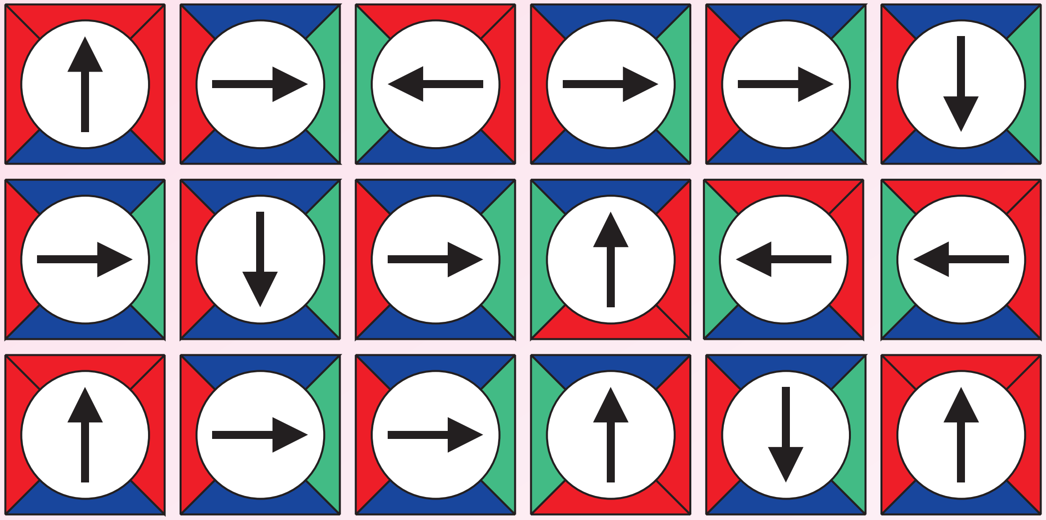
SNAKES is a tile set with some interesting (and useful) properties.

SNAKES are Wang tiles with an arrow printed on them. It points to one of the four neighbors of the tile:

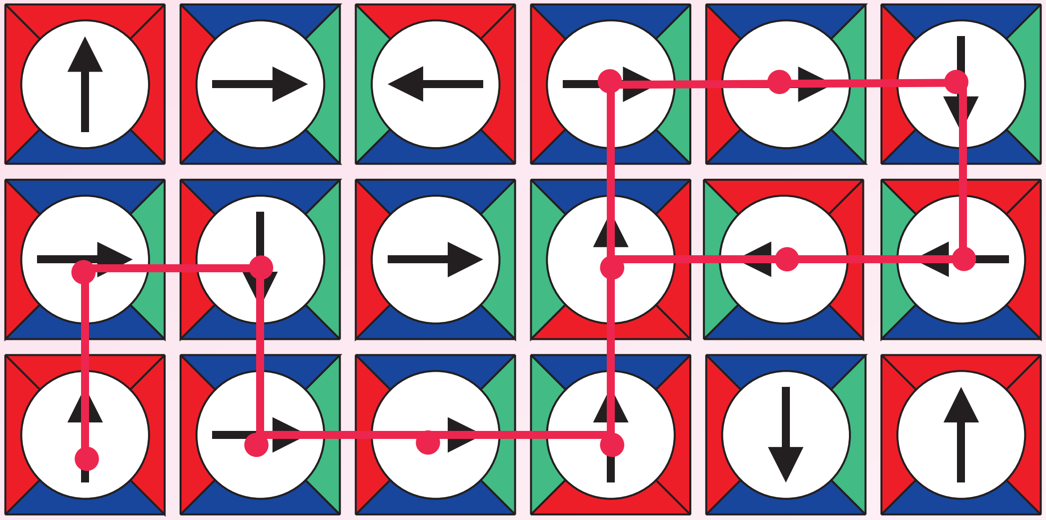


Such tiles with arrows are called **directed tiles**.

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:

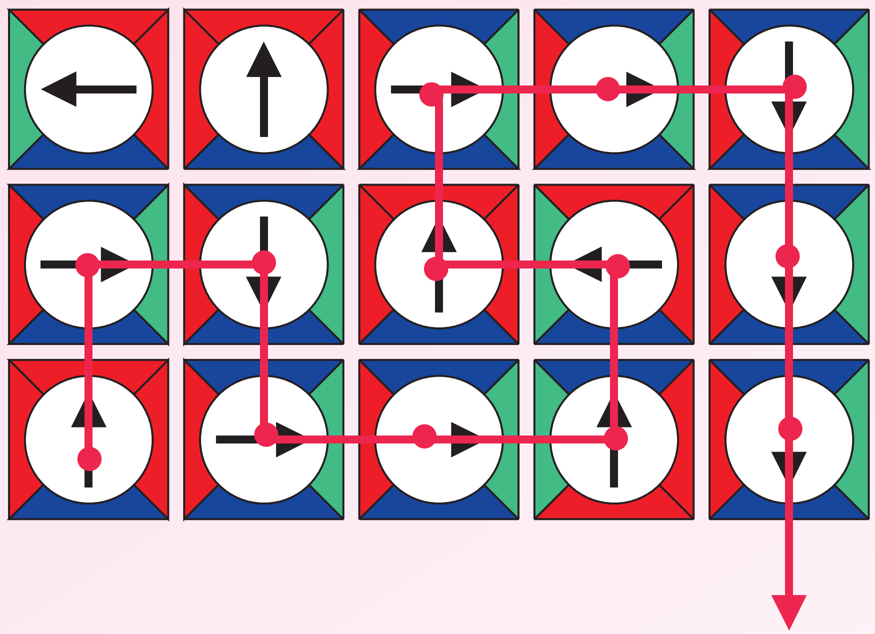


Given a configuration (valid tiling or not!) and a starting position, the arrows specify a path on the plane. Each position is followed by the neighboring position indicated by the arrow of the tile:



The path may enter a loop…

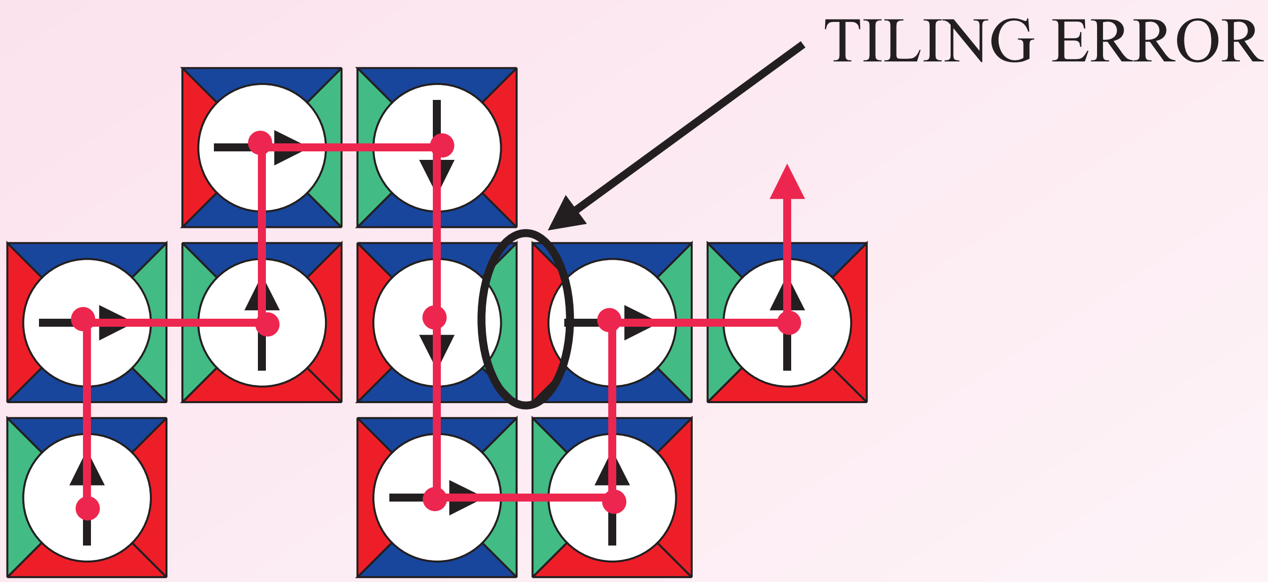
Given a configuration (valid tiling or not!) and a starting position, the arrows specify a path on the plane. Each position is followed by the neighboring position indicated by the arrow of the tile:



...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:

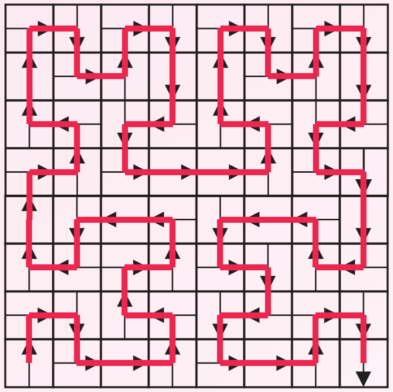
(1) Either there is a tiling error at some tile along the path,



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 at some tile along the path,

(2) or the path is a **plane-filling path**: 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:

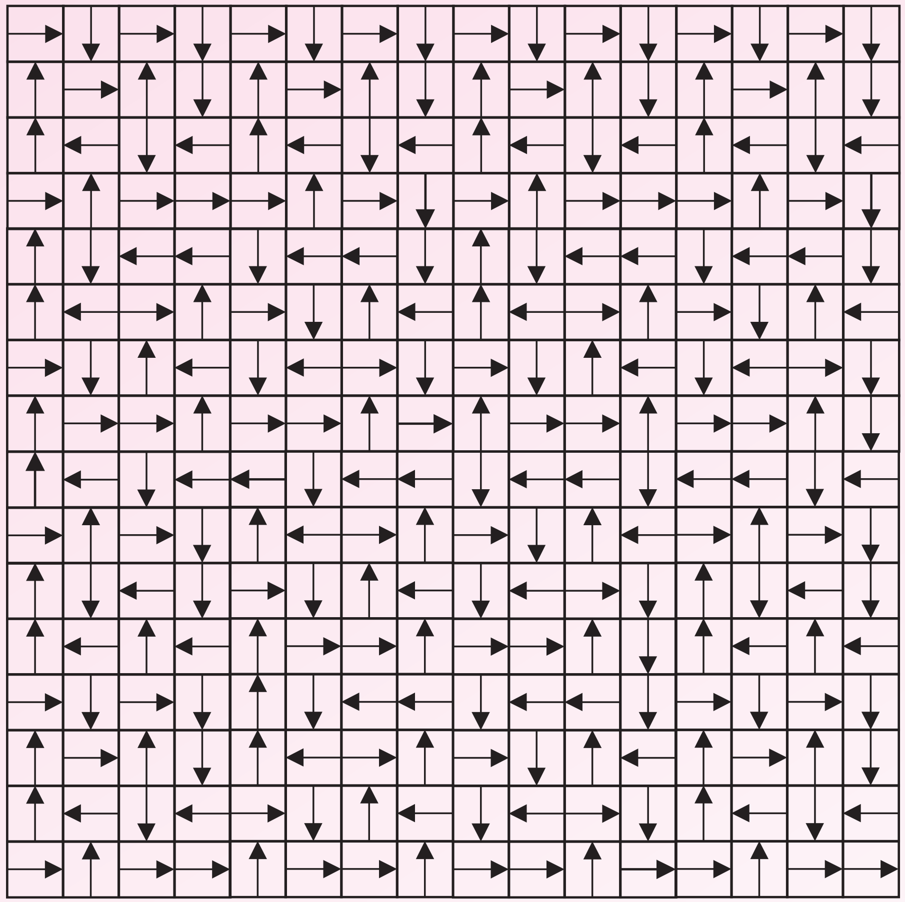
(1) Either there is a tiling error at some tile along the path,

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

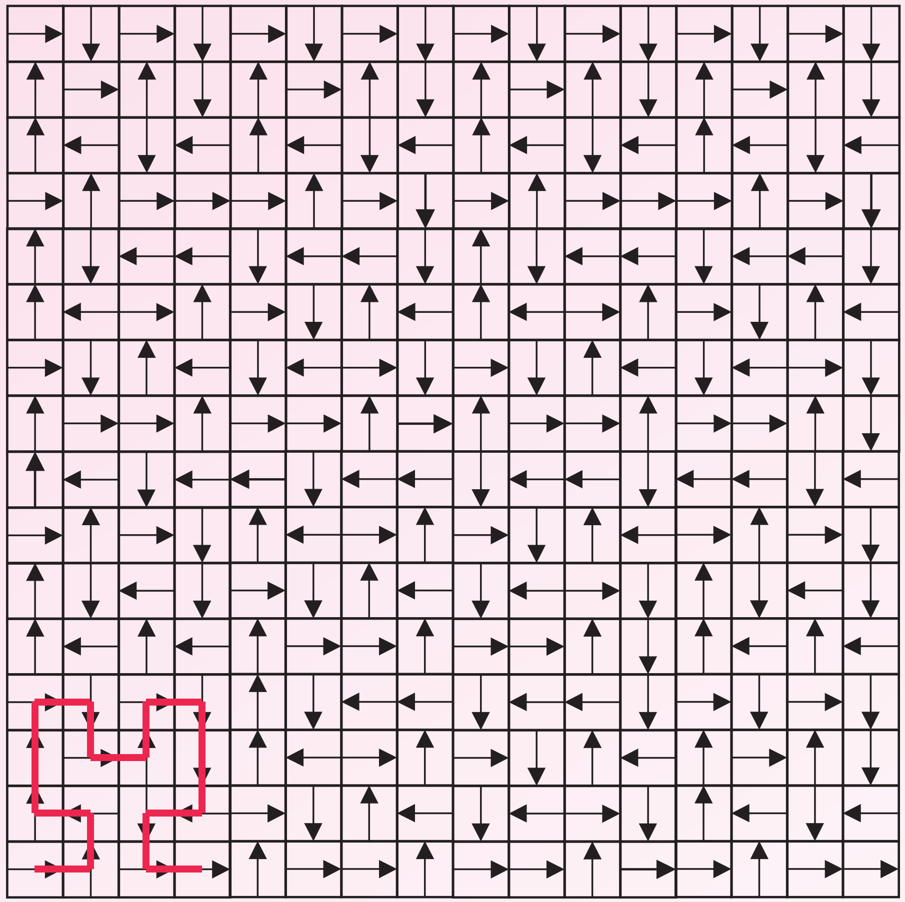
Note that the tiling may be invalid outside path P, yet the path is forced to snake through larger and larger squares.

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

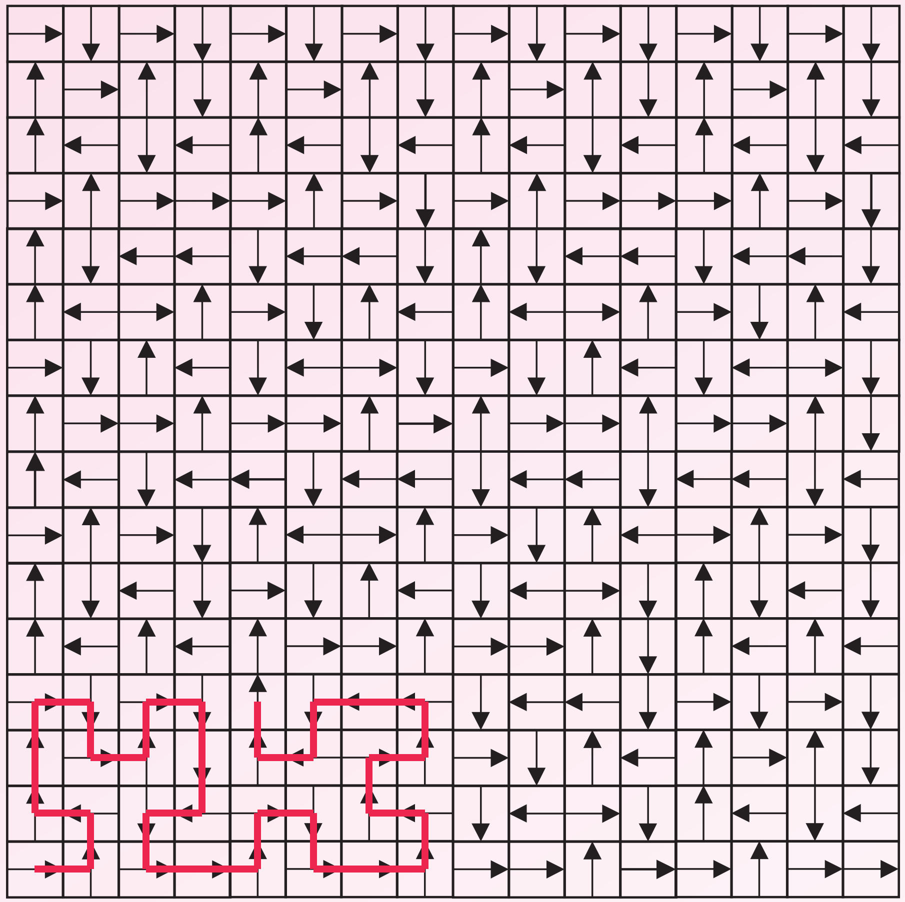
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



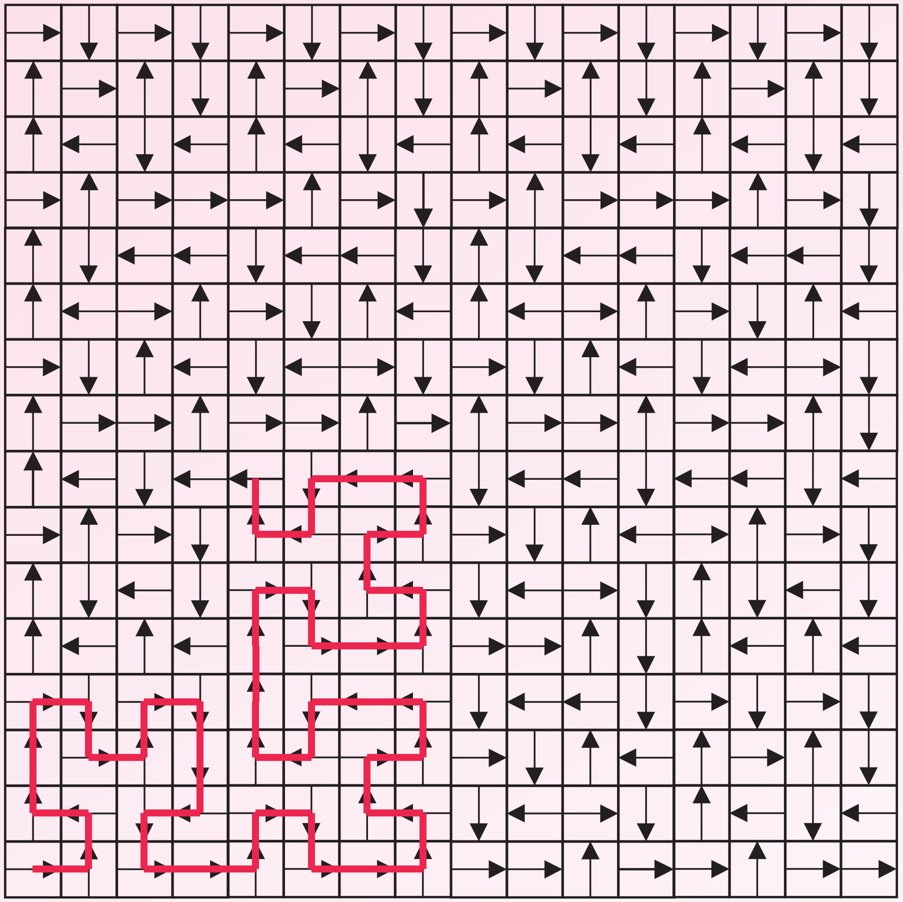
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



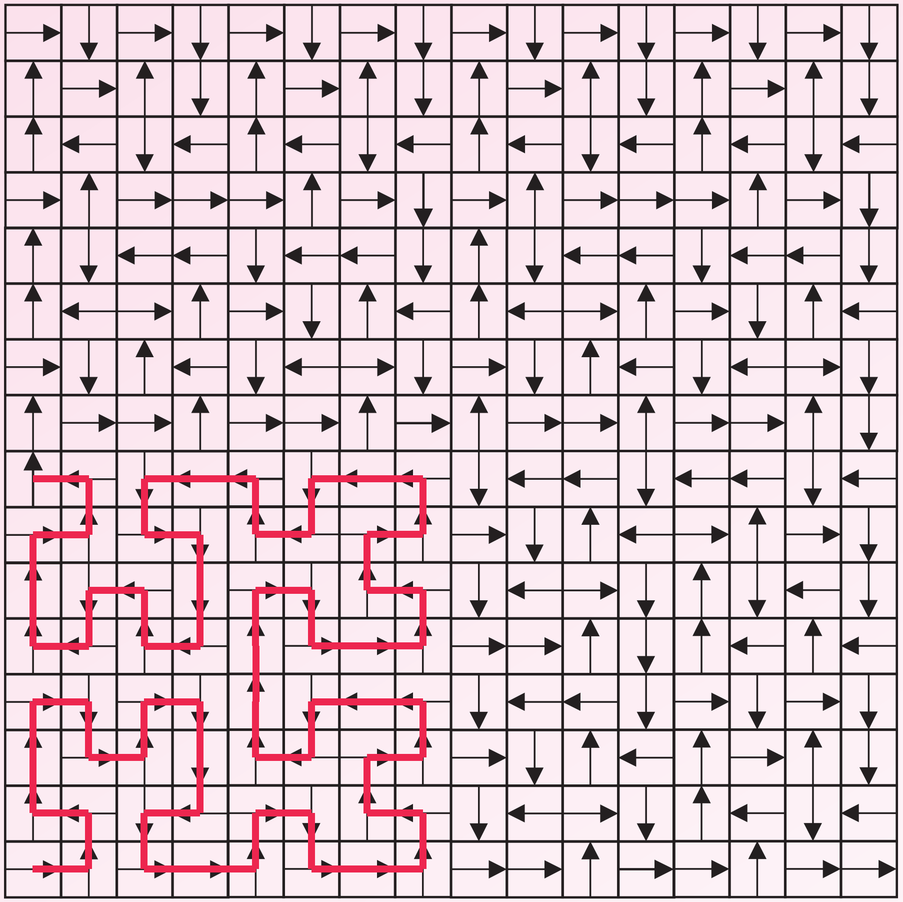
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



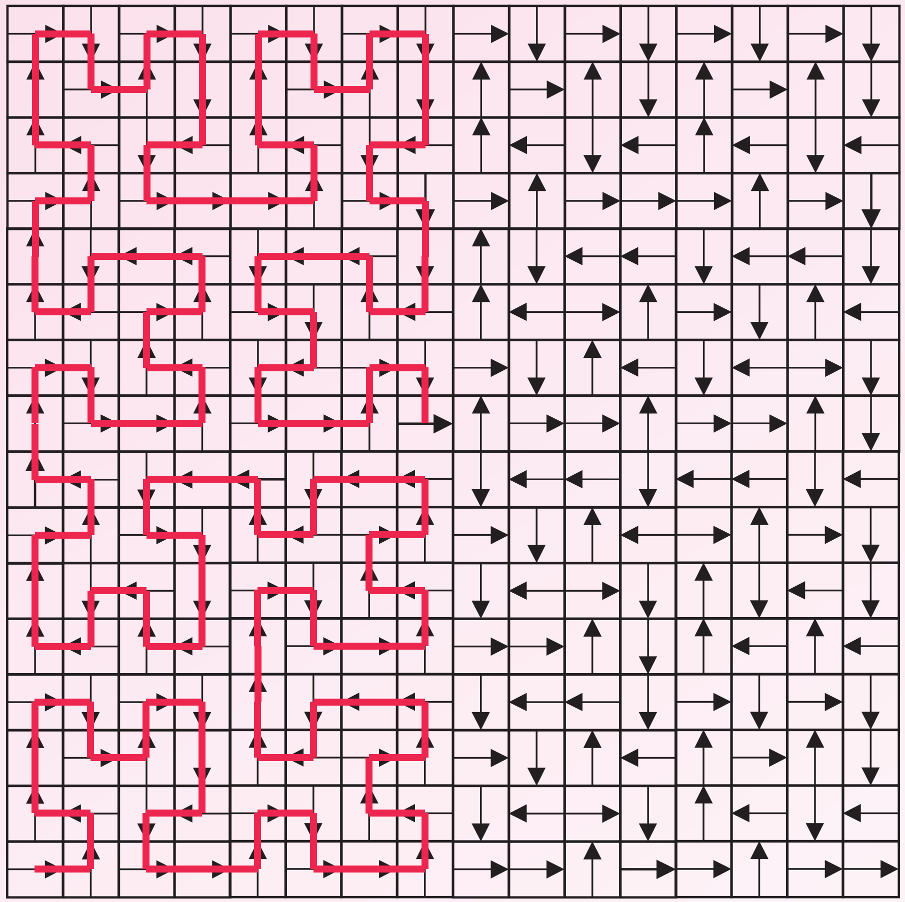
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



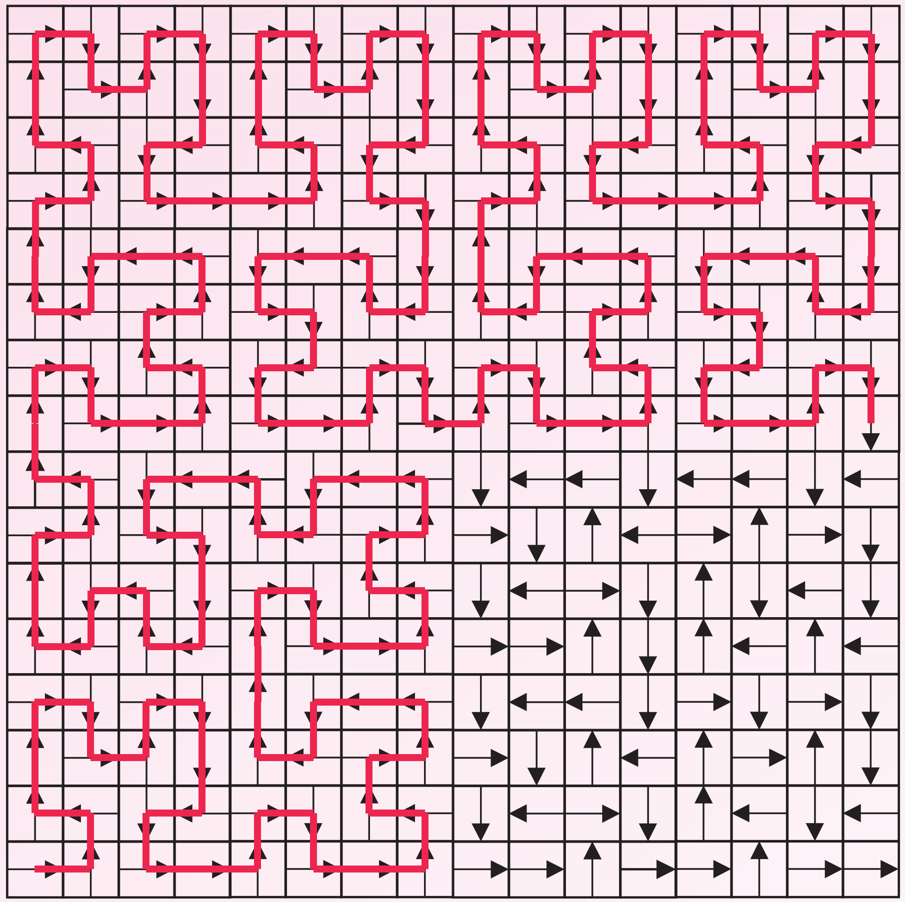
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



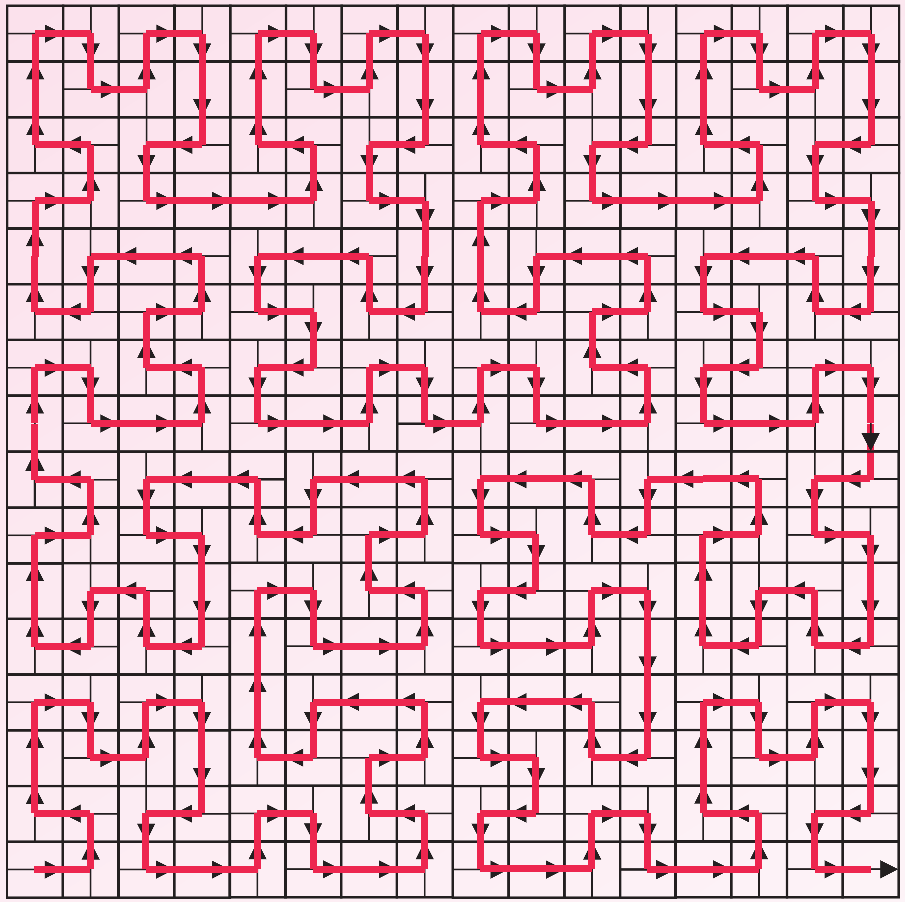
The paths that SNAKES forces when no tiling error is encountered have the shape of the well known plane-filling Hilbert-curve



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

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

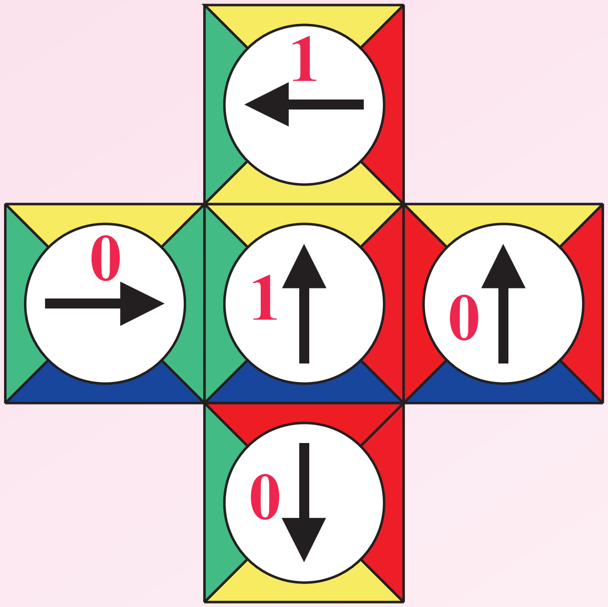
The state set of the CA is

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

(Each snake tile is attached a red bit.)

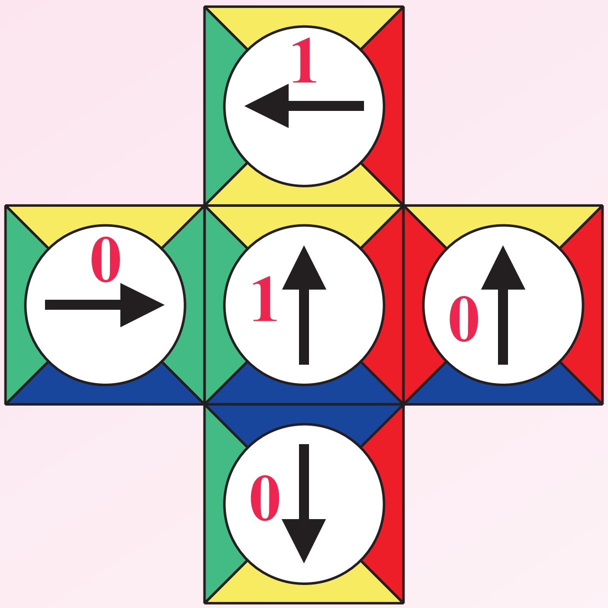
The local rule checks whether the tiling is valid at the cell:

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



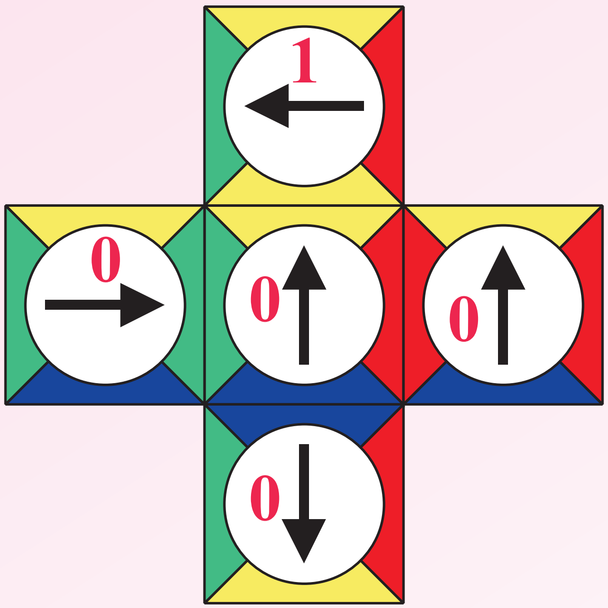
The local rule checks whether the tiling is valid at the cell:

- 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.

1 1 1 1 1	0 0 0 0 0
1 1 1 1	0 0 0 0
1 1 1 1 1	0 0 0 0 0

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 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.

0 0 0 0 0	0 0 0 0 0
0 0 0 0	0 0 0 0
0 0 0 0 0	0 0 0 0 0

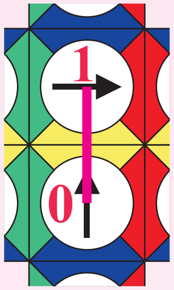
All cells are active because the tilings are correct. This means that all bits in both configurations become 0. So the two configurations become identical. The CA is not injective.

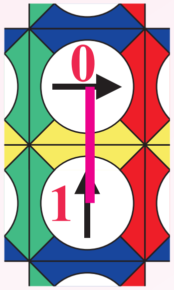
Snake XOR is injective on periodic configurations:

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 $\vec{p}_1 \in \mathbb{Z}^2$.

Because c and d have identical successors:

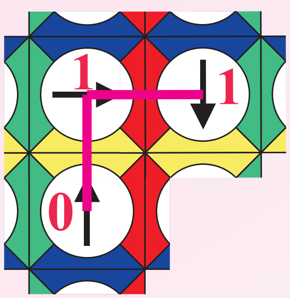
- The cell in position $\vec{p}_1$ must be active, that is, the Snakes tiling is valid in position $\vec{p}_1$.
- The bits stored in the next position $\vec{p}_2$ (indicated by the direction) are different in c and d.

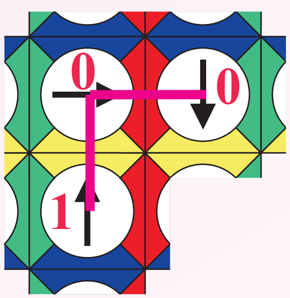




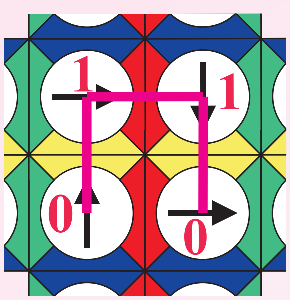
We repeat the reasoning in position $\vec{p}_2$:

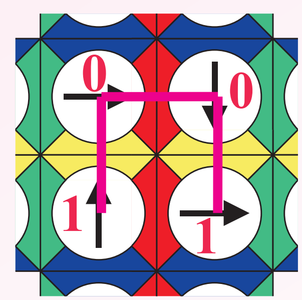
- The SNAKES tiling is valid in position $\vec{p}_2$.
- The bits stored in the next position $\vec{p}_3$ are different in c and d.



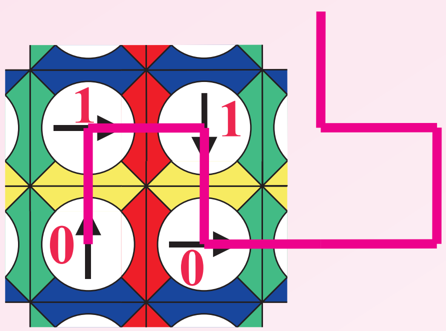


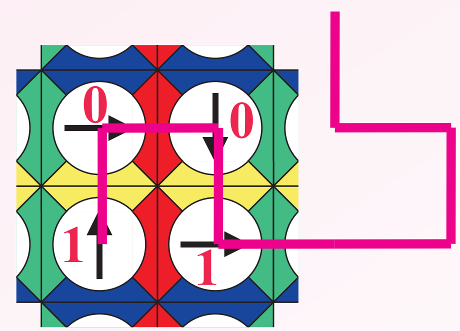
The same reasoning can be repeated over and over again. The positions $\vec{p}_1, \vec{p}_2, \vec{p}_3, \ldots$ form a path that follows the arrows on the tiles. There is no tiling error at any tile on this path.





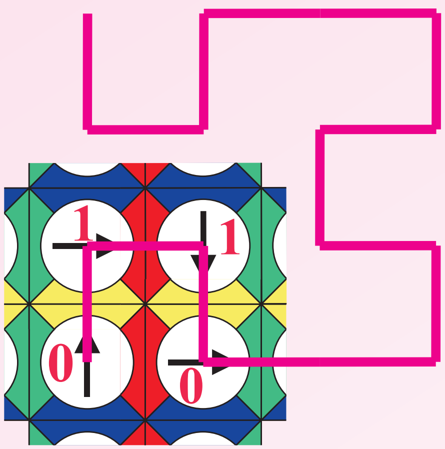
The same reasoning can be repeated over and over again. The positions $\vec{p}_1, \vec{p}_2, \vec{p}_3, \ldots$ form a path that follows the arrows on the tiles. There is no tiling error at any tile on this path.

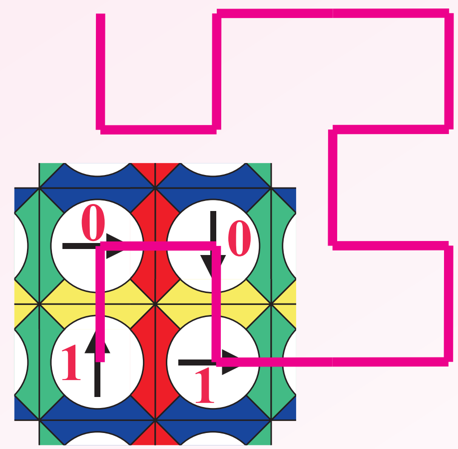




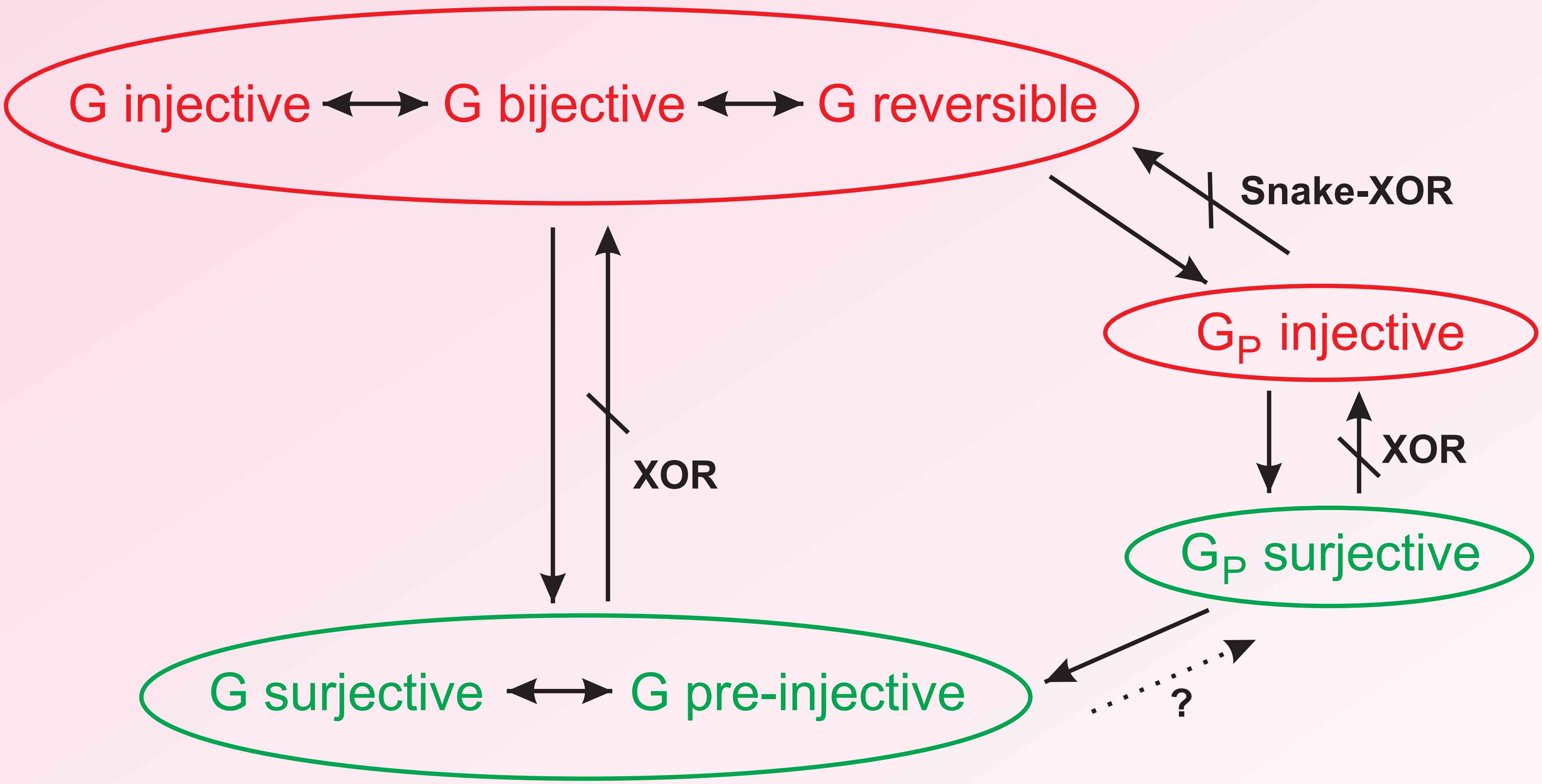
The same reasoning can be repeated over and over again. The positions $\vec{p}_1, \vec{p}_2, \vec{p}_3, \ldots$ form a path that follows the arrows on the tiles. There is no tiling error at any tile on this path.

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





In 2D



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

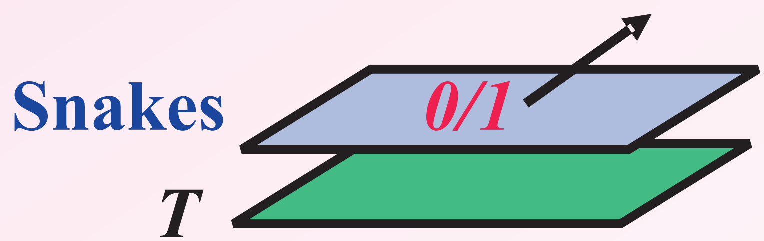
Second application of Snakes: It is undecidable to determine if a given two-dimensional CA is reversible.

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.

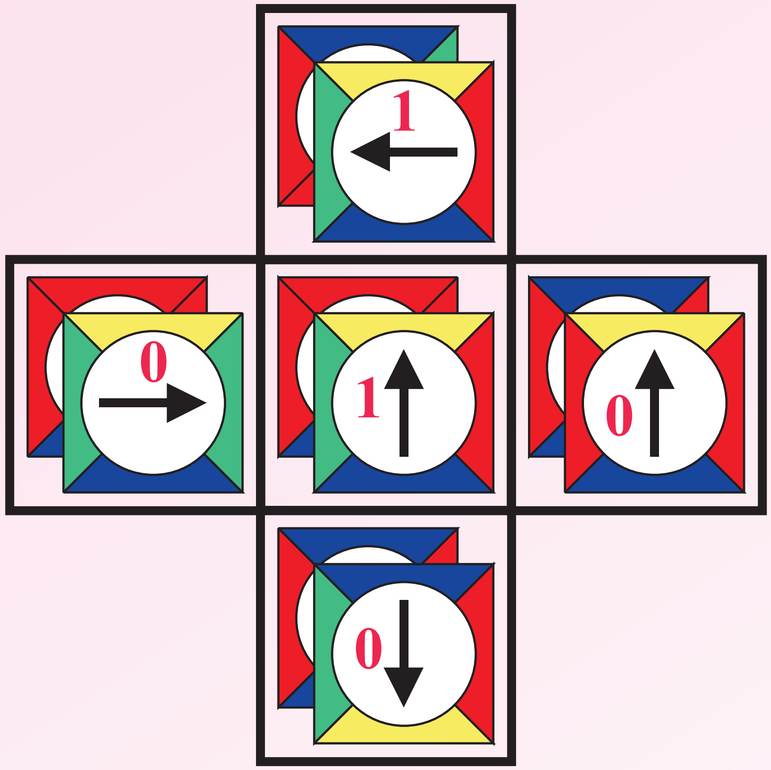
For any given tile set T we construct a CA with the state set

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



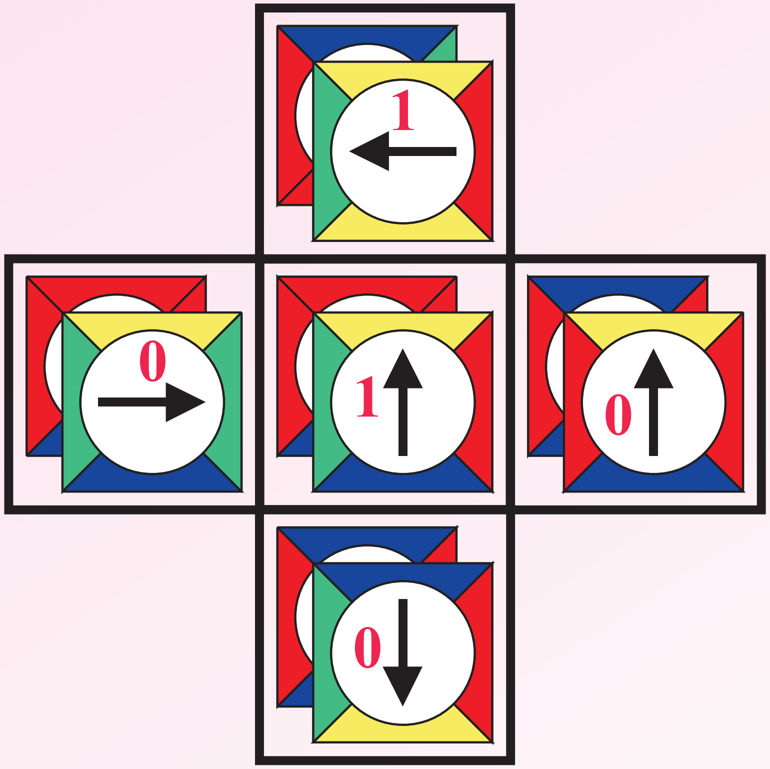
The local rule is analogous to **Snake XOR** with the difference that the correctness of the tiling is checked in both tile layers:

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



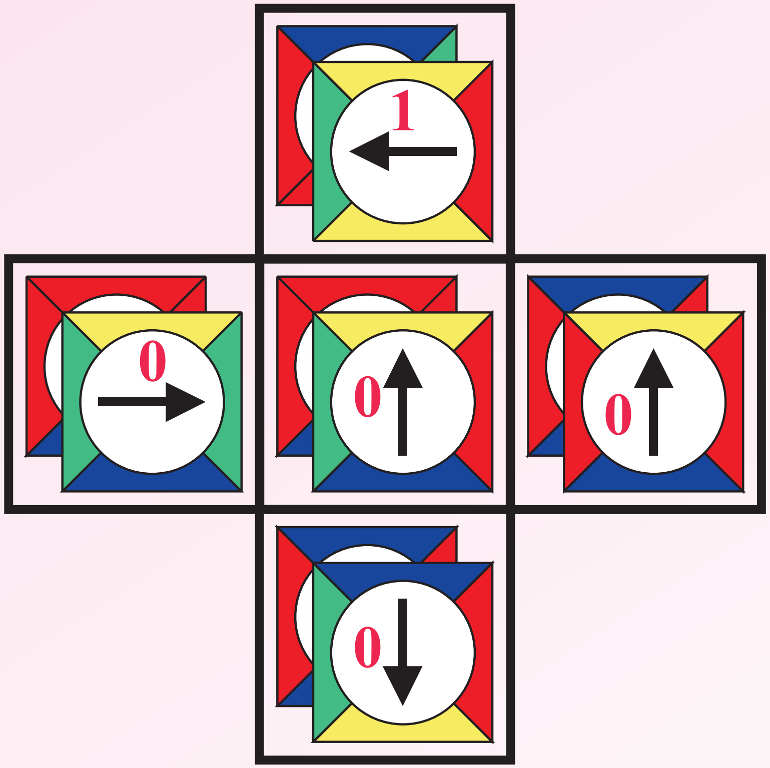
The local rule is analogous to **Snake XOR** with the difference that the correctness of the tiling is checked in both tile layers:

- 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.

(T tiles $\Longrightarrow$ CA not reversible) 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.

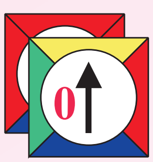
All cells are active because the tilings are correct. This means that all bits in both configurations become 0. So the two configurations become identical. The CA is not injective.

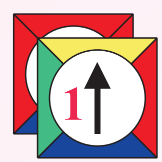
(T tiles $\implies$ CA not reversible) If a valid tiling of the plane exists then we can construct two different configurations of the CA that have the same image under G. The Snakes and the T layers of the configurations form the same valid tilings of the plane. In one of the configurations all bits are 0, and in the other configuration all bits are 1.

0 0 0 0 0   0 0 0 0 0
0 0 0 0   0 0 0 0
0 0 0 0 0   0 0 0 0 0

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.

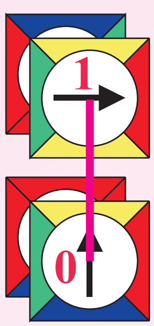
(T tiles $\Longleftarrow$ CA not reversible) Conversely, assume that the CA is not injective. Let c and d be two different configurations with the same successor. Since only bits may change, c and d must have identical SNAKES and T layers. So they must have different bits 0 and 1 in some position $\vec{p}_1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

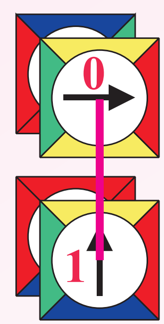




Because c and d have identical successors:

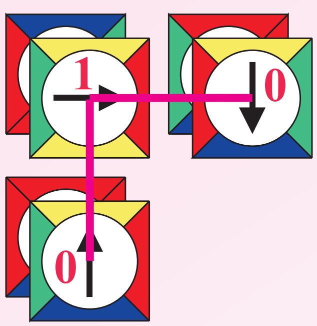
- The cell in position $\vec{p}_1$ must be active, that is, the Snakes and T tilings are both valid in position $\vec{p}_1$.
- The bits stored in the next position $\vec{p}_2$ (indicated by the direction) are different in c and d.

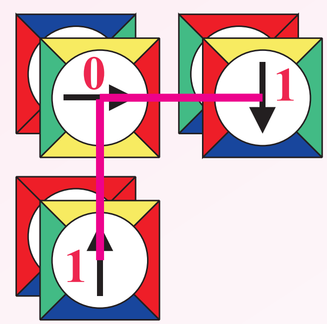




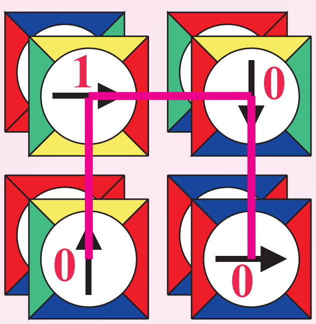
We repeat the reasoning in position $\vec{p}_2$:

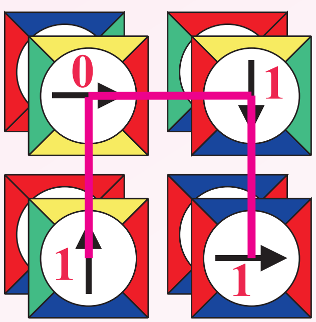
- The SNAKES and T tilings are valid in position $\vec{p}_2$.
- The bits stored in the next position $\vec{p}_3$ are different in c and d.



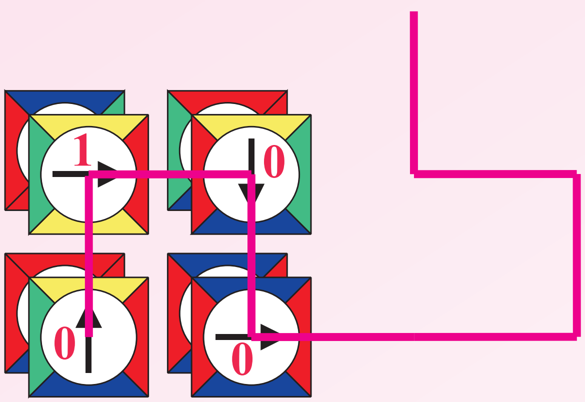


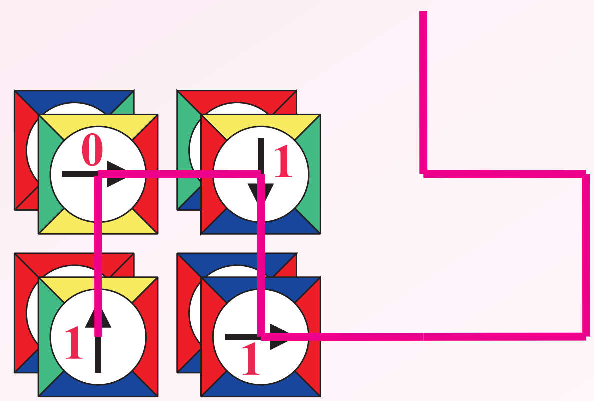
The same reasoning can be repeated over and over again. The positions $\vec{p}_1, \vec{p}_2, \vec{p}_3, \ldots$ form a path that follows the arrows on the tiles. There is no tiling error at any tile on this path so the special property of Snakes forces the path to cover arbitrarily large squares.



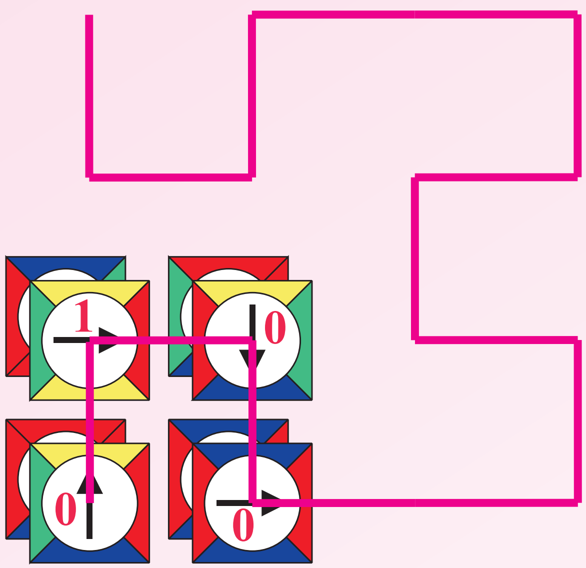


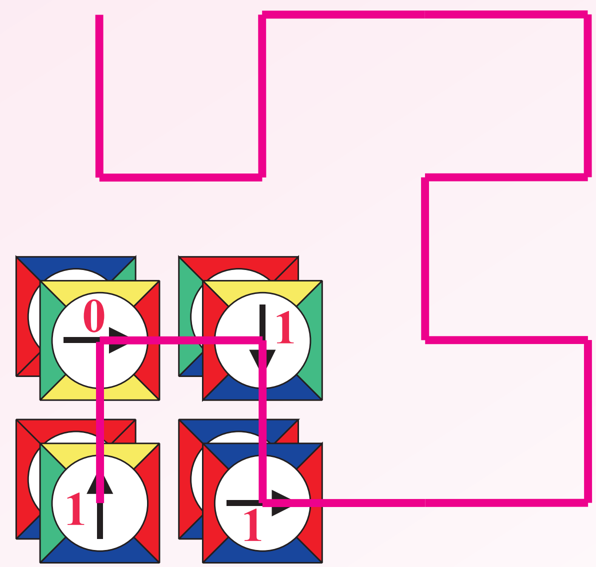
The same reasoning can be repeated over and over again. The positions $\vec{p}_1, \vec{p}_2, \vec{p}_3, \ldots$ form a path that follows the arrows on the tiles. There is no tiling error at any tile on this path so the special property of Snakes forces the path to cover arbitrarily large squares.





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.





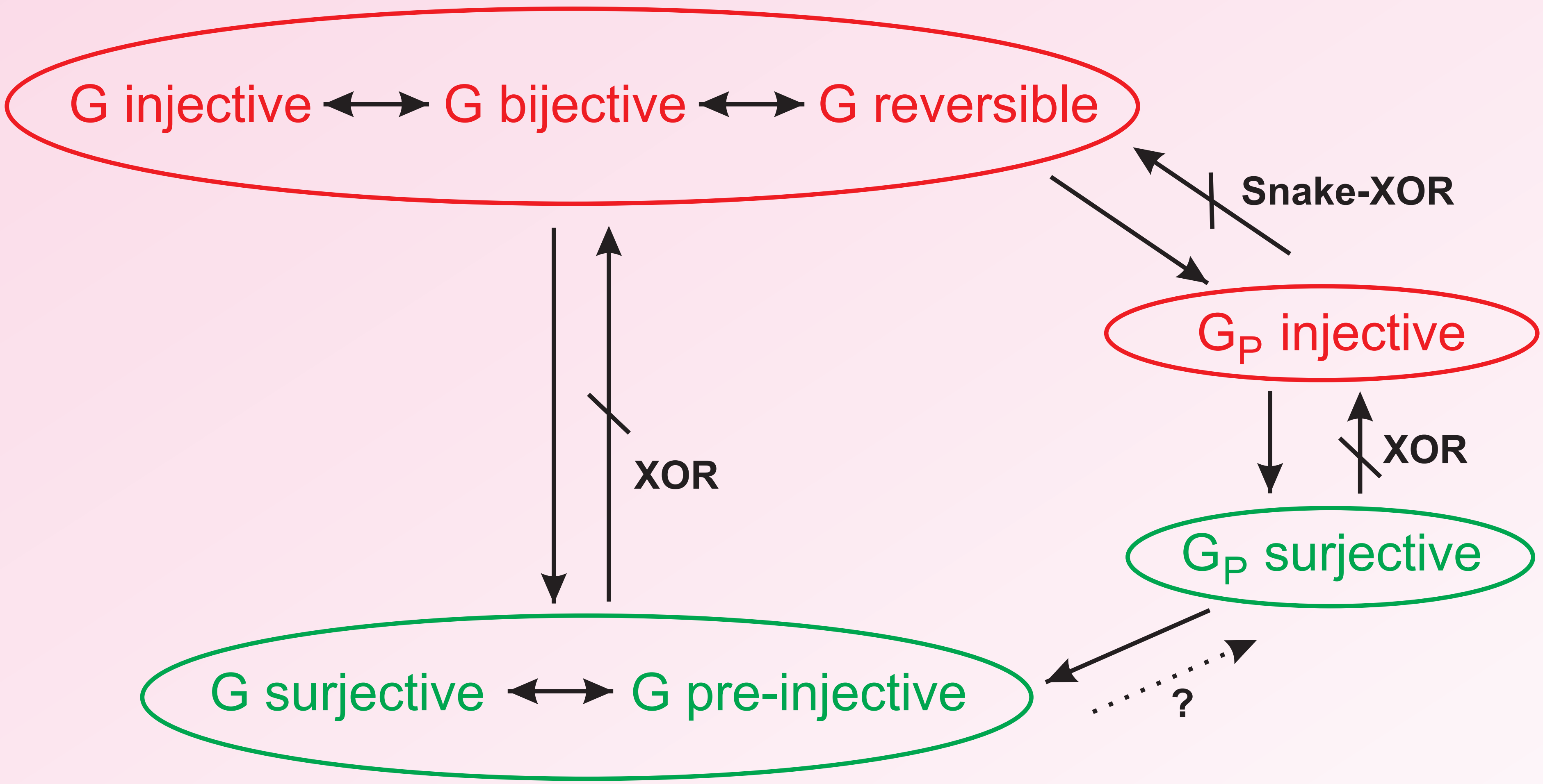
Hence T admits tilings of arbitrarily large squares, and consequently a tiling of the infinite plane. $\square$

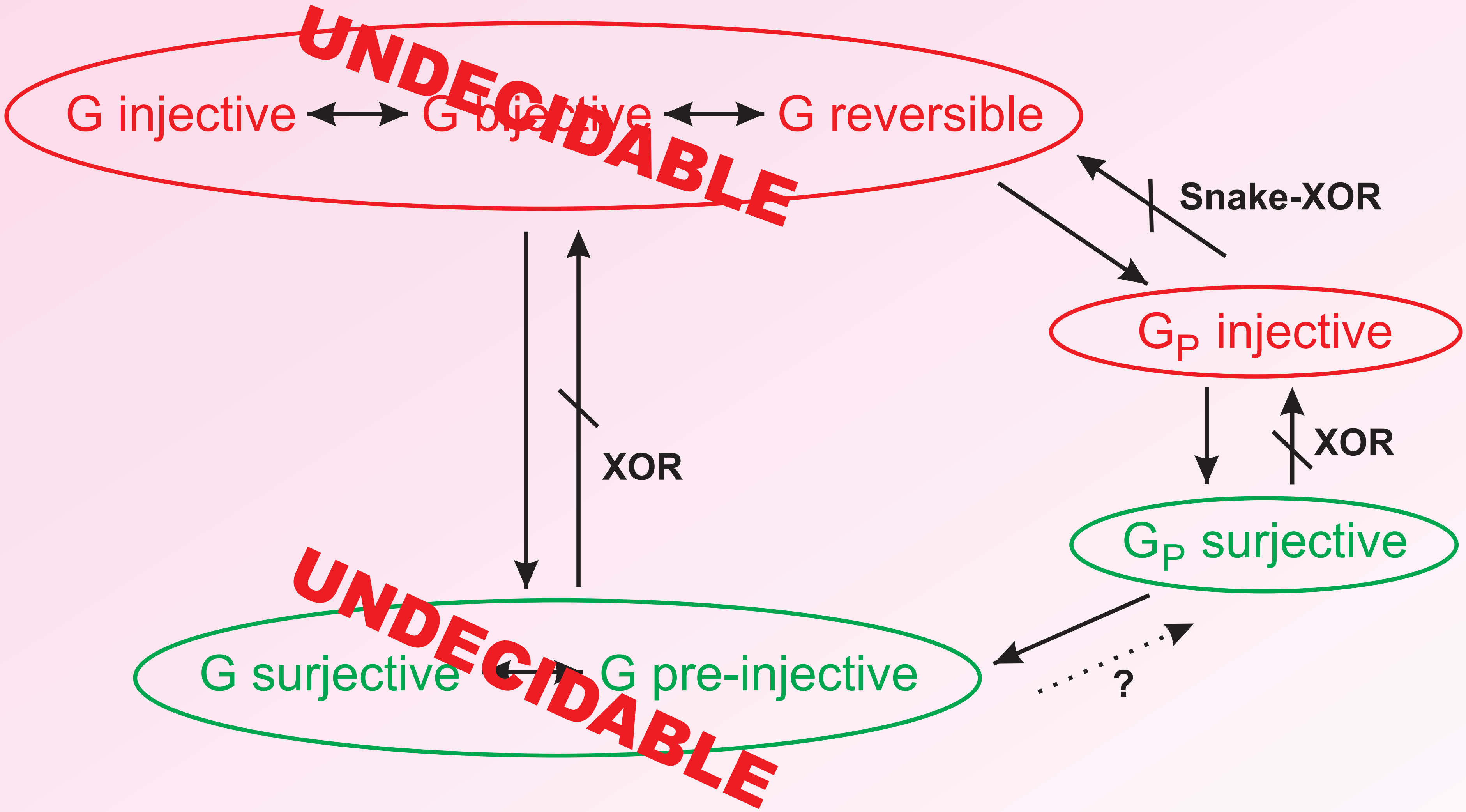
Theorem: It is undecidable whether a given two-dimensional CA is injective.

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

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
?



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

Both problems are semi-decidable in one direction:

Injectivity is semi-decidable: Enumerate all CA G one-by one and check if G is the inverse of the given CA. Halt once (if ever) the inverse is found.

Non-surjectivity is semi-decidable: Enumerate all finite patterns one-by-one and halt once (if ever) an orphan is found.

Undecidability of injectivity 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 injectivity 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.

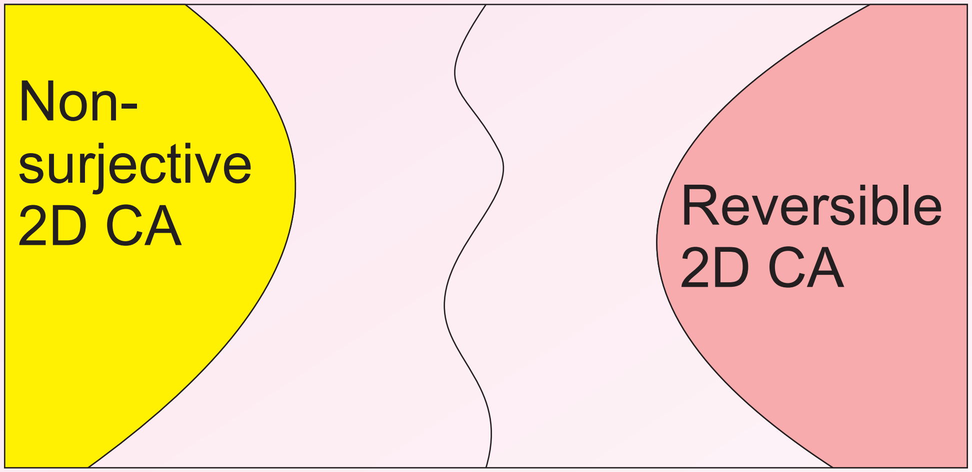
So while the smallest known orphan for Game-Of-Life is pretty big (92 cells), this pales in comparison with some other CA.

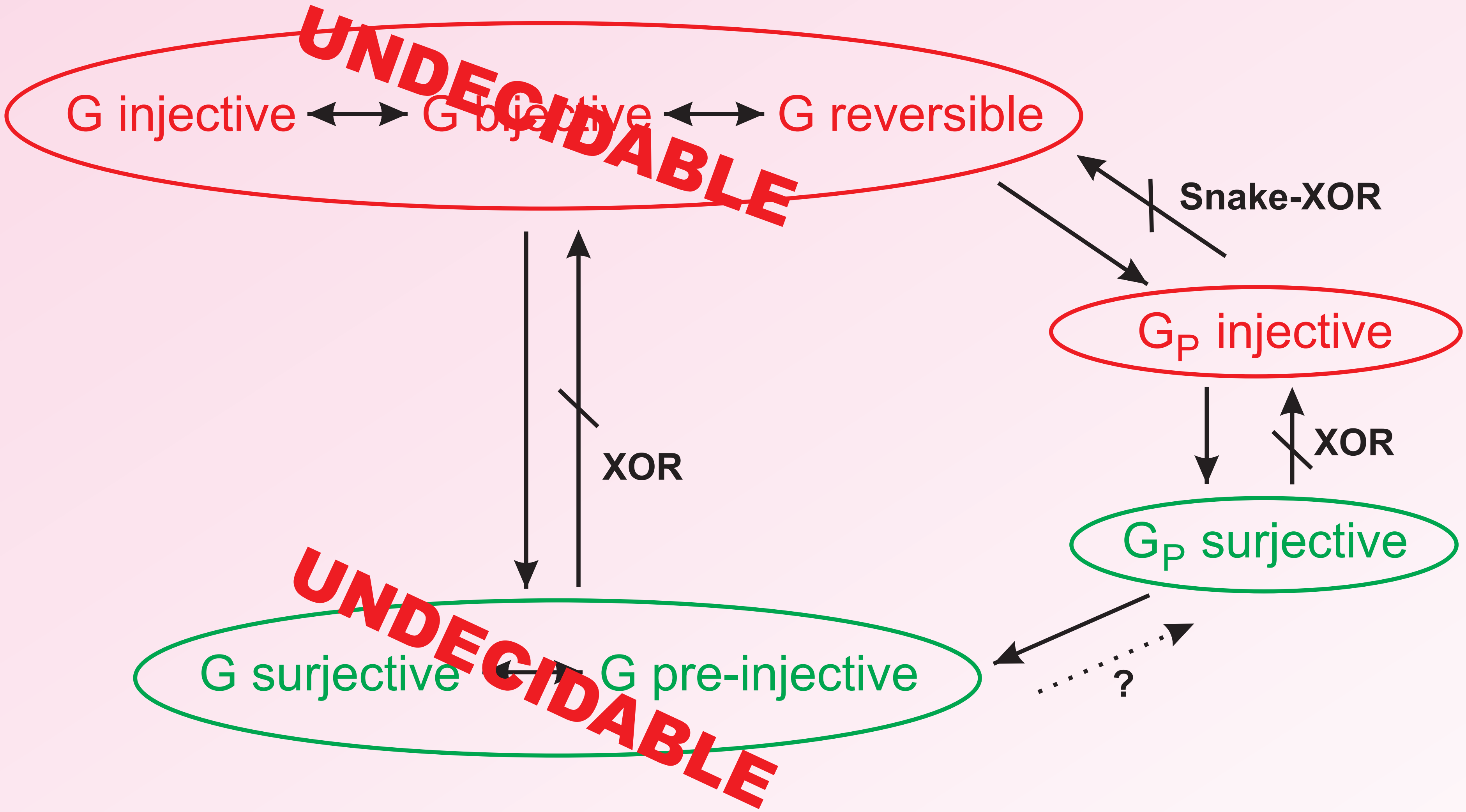
The undecidability proofs for reversibility and surjectivity can be merged into

Theorem: The classes of

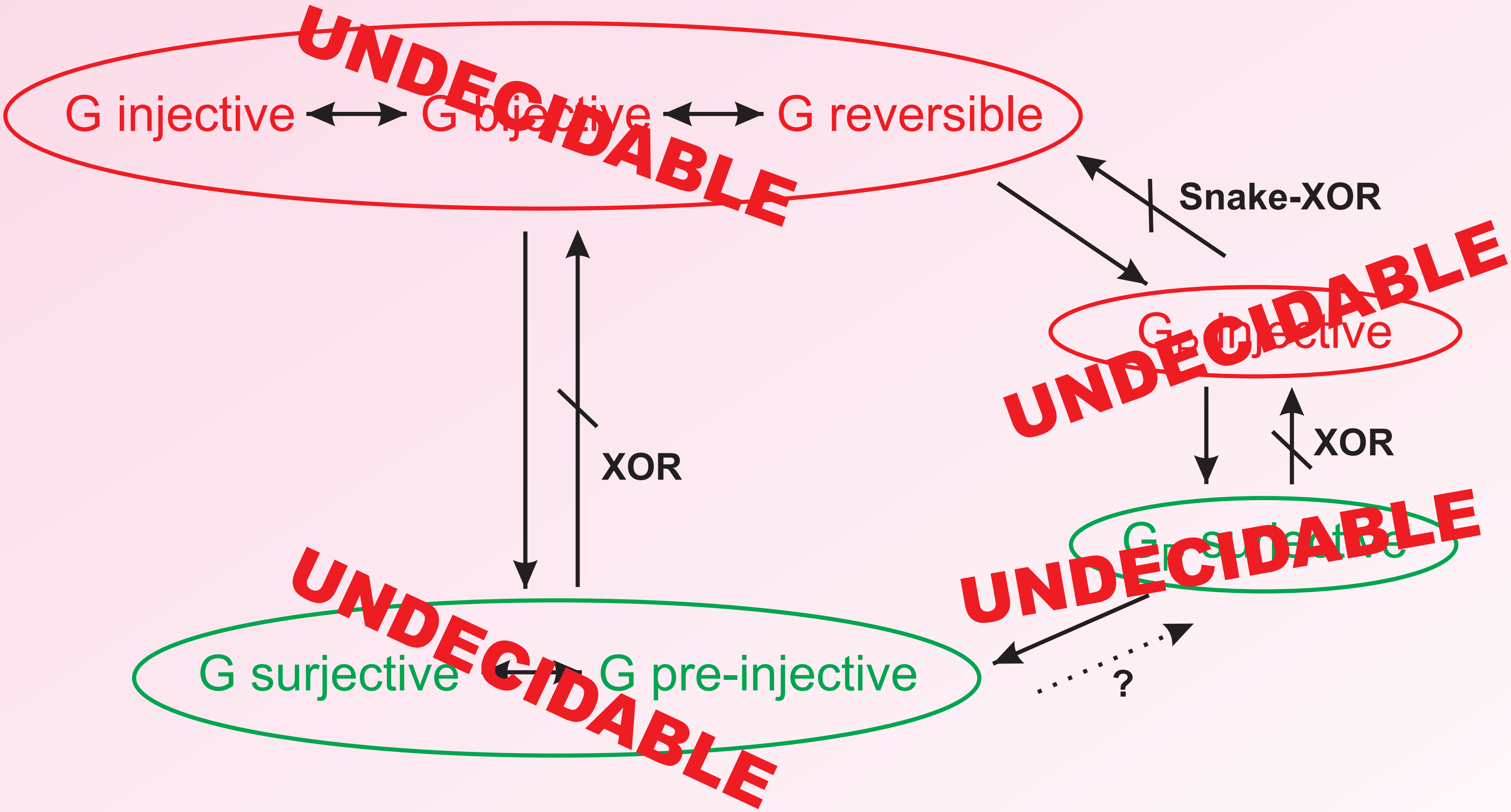
- Reversible 2D CA
- Non-surjective 2D CA

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

Some challenging open problems

We have not (yet) managed to prove everything about CA.

Some challenging open problems

We have not (yet) managed to prove everything about CA.

- Universality of elementary rule 54.

Some challenging open problems

We have not (yet) managed to prove everything about CA.

- Universality of elementary rule 54.
- "G surjective $\stackrel{?}{\Longrightarrow}$ G_P surjective" among 2D CA

Some challenging open problems

We have not (yet) managed to prove everything about CA.

- Universality of elementary rule 54.
- "G surjective $\overset{?}{\Longrightarrow}$ G_P surjective" among 2D CA
- Are temporally periodic configurations dense on surjective CA ? (=does every finite pattern occur in some temporally periodic configuration?)

Some challenging open problems

We have not (yet) managed to prove everything about CA.

- Universality of elementary rule 54.
- "G surjective $\stackrel{?}{\Longrightarrow}$ G_P surjective" among 2D CA
- Are temporally periodic configurations dense on surjective CA ? (=does every finite pattern occur in some temporally periodic configuration?)
- Is it decidable if a given CA is **positively expansive** ? (CA is positively expansive if the view through some finite observation window on orbit $c, G(c), \ldots$ uniquely identifies the initial configuration c.)

Some challenging open problems

We have not (yet) managed to prove everything about CA.

- Universality of elementary rule 54.
- "G surjective $\stackrel{?}{\Longrightarrow}$ G_P surjective" among 2D CA
- Are temporally periodic configurations dense on surjective CA ? (=does every finite pattern occur in some temporally periodic configuration?)
- Is it decidable if a given CA is **positively expansive** ? (CA is positively expansive if the view through some finite observation window on orbit $c, G(c), \ldots$ uniquely identifies the initial configuration c.)

Any solutions are welcome at STACS'17

Big thanks to everyone for listening…

…now let's go to the welcome reception.