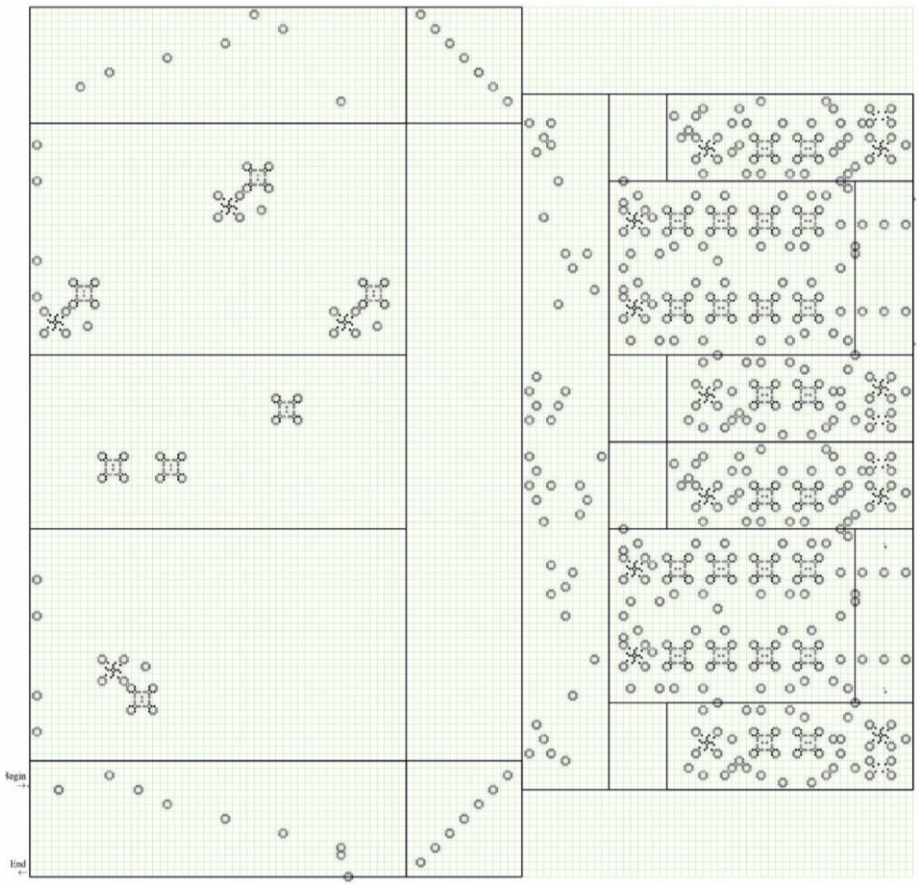
森田憲
Kenichi Morita
(1949-2025)

A Pioneer of
Reversible Computation
and
Cellular Automata

In July we received sad news: our dear colleague, professor Kenichi Morita, had passed away



Kenichi Morita did groundbreaking research in reversible cellular automata, reversible counter machines, reversible Turing machines, and in reversible computing in general.

Morita's 1989 article with M. Harao was a breakthrough that solved a major open problem: The article proved that one-dimensional reversible cellular automata (RCA) can be computationally universal.

In this talk I review this Morita-Harao construction.

758

THE TRANSACTIONS OF THE IEICE, VOL. E **72**, NO. **6** JUNE **1989**

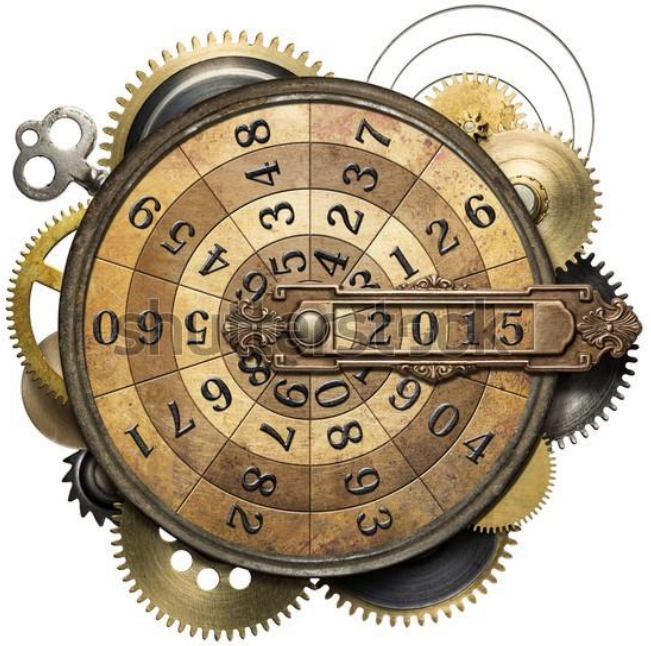
PAPER

Computation Universality of One-Dimensional Reversible (Injective) Cellular Automata

Kenichi MORITA† *and* Masateru HARAO††, *Members*

SUMMARY A reversible cellular automaton (CA) is a "backward deterministic" CA, i. e., every configuration of it has at most one predecessor. Toffoli showed that a two-dimensional reversible cellular automaton is computation universal. He posed an open problem whether a one-dimensional reversible CA is computation universal. In this paper, we solve this problem affirmatively. This computation universal). Previously, by improving Bennett's result, Morita et al.[4]. showed that a 1-tape 2-symbol reversible Turing machine is computation universal. In this paper, we show that any given 1-tape reversible Turing machine can be simulated by a one-

So let us take our time machine and return 36 years back in time.



In 1989 much less is known about reversible cellular automata than today. Recall that a *d*-dimensional CA

$$G : A^{\mathbb{Z}^d} \to A^{\mathbb{Z}^d}$$

is called reversible if it is

- bijective and
- the inverse function G^{-1} is also a cellular automaton.

A simple compactness argument reveals that if G is bijective then the inverse G^{-1} is automatically a cellular automaton (with potentially a much larger neighborhood than G).

It was also known that every injective CA is surjective, and hence bijective. Actually two very different proofs of this fact were known:

- Using the Garden-of-Eden theorem:

$$G \text{ injective} \Rightarrow G \text{ pre-injective} \Rightarrow G \text{ surjective}$$

- Through periodic configurations:

$$G \text{ injective} \Rightarrow G_P \text{ injective} \Rightarrow G_P \text{ surjective} \Rightarrow G \text{ surjective}$$

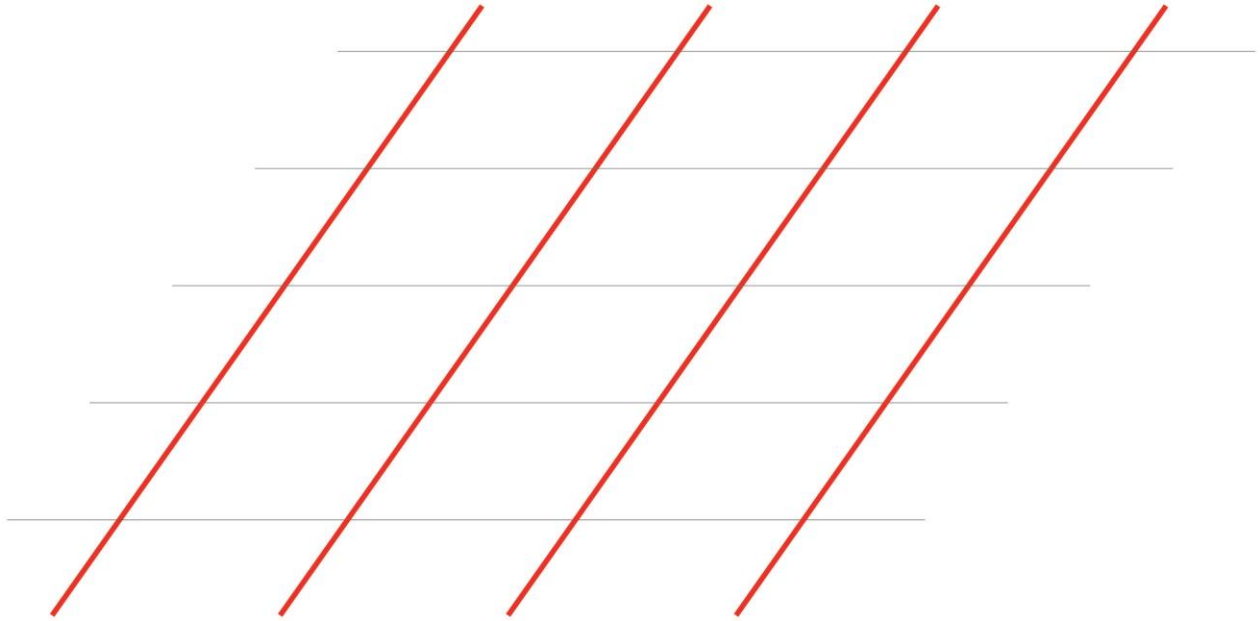
where G_P is the restriction of G on (strongly) periodic configurations.

Thus every injective CA function is reversible.

It was also known that two- and higher dimensional RCA can by computationally universal. Tommaso Toffoli had shown in 1977 how any *d*-dimensional (irreversible) CA can be simulated by a *(d+1)*-dimensional RCA.

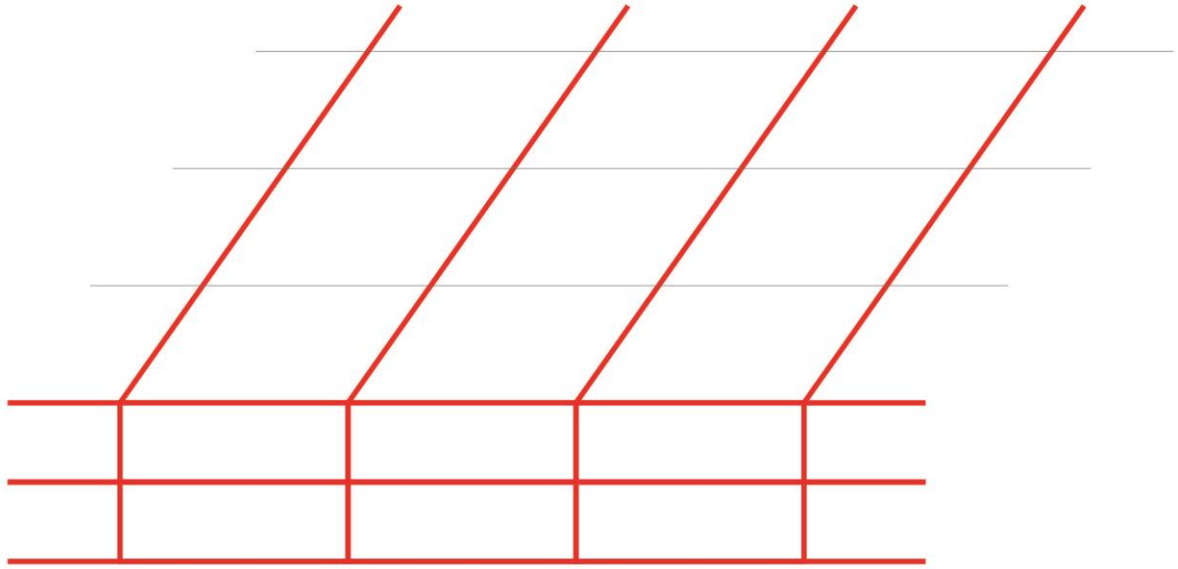
As one-dimensional universal CA were known, we directly get two-dimensional universal RCA from Toffoli's construction.

Example: Let *F* be a universal one-dimensional CA with the state set *A={0,1}* where 0 is a quiescent state. (For example, F could be the elementary CA number 110.)
Here is a reversible two-dimensional CA that simulates F step-by-step:

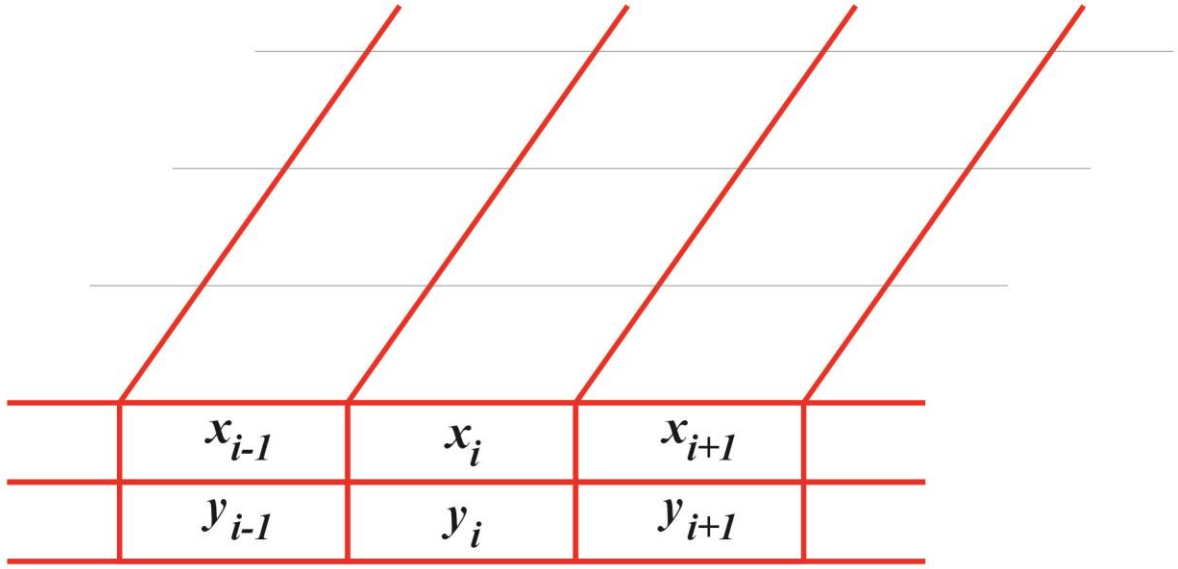


Vertical strips of two-dimensional configurations will operate as one-dimensional configurations.

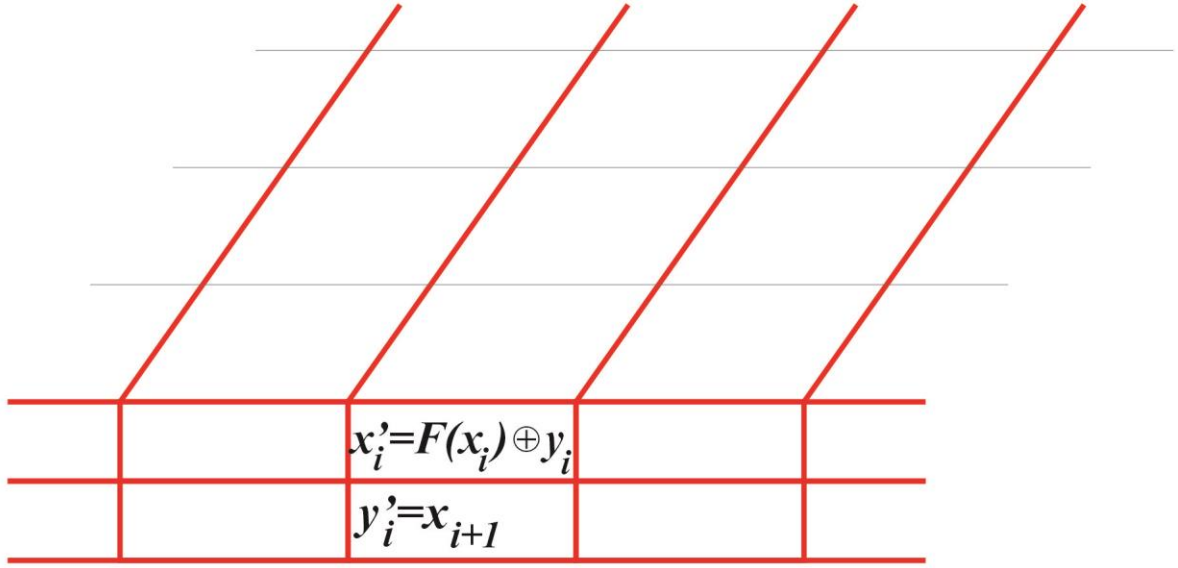
Example: Let *F* be a universal one-dimensional CA with the state set *A={0,1}* where 0 is a quiescent state. (For example, F could be the elementary CA number 110.) Here is a reversible two-dimensional CA that simulates F step-by-step:



Each strip has two tracks. Both tracks store states from *A={0,1}* . The second track is used to collect computation history.



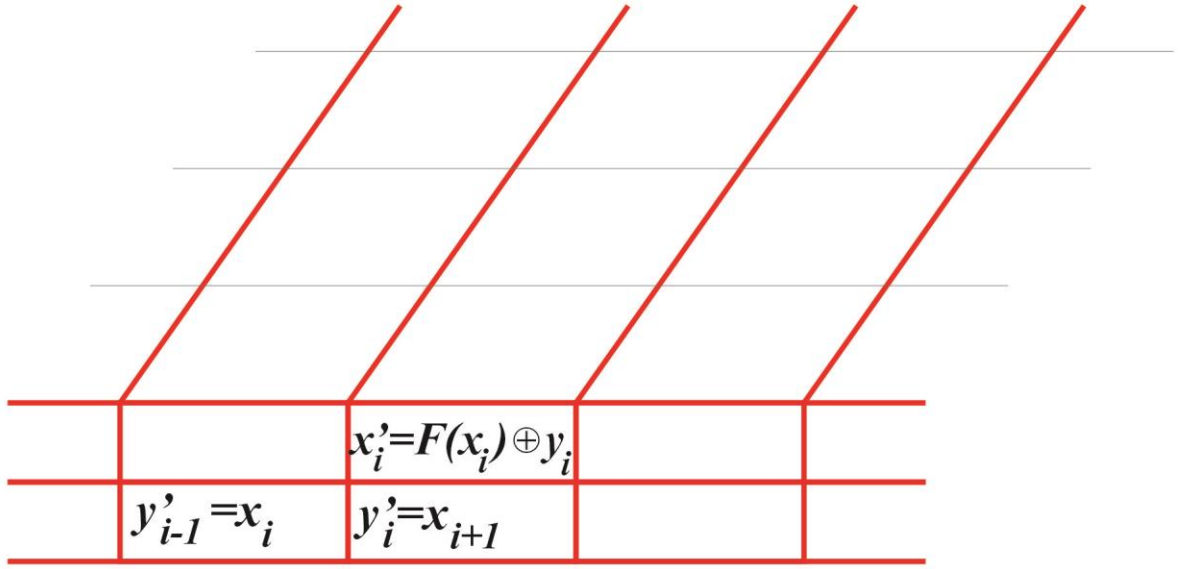
Let $x_i \in \{0,1\}^{\mathbb{Z}}$ be the sequences on the upper layer, and $y_i \in \{0,1\}^{\mathbb{Z}}$ on the lower layer, for $i \in \mathbb{Z}$. (So in the drawing above the symbol in the cross-section represents the entire bi-infinite vertical sequence of bits.)



The reversible two-dimensional local rule replaces x_i and y_i by

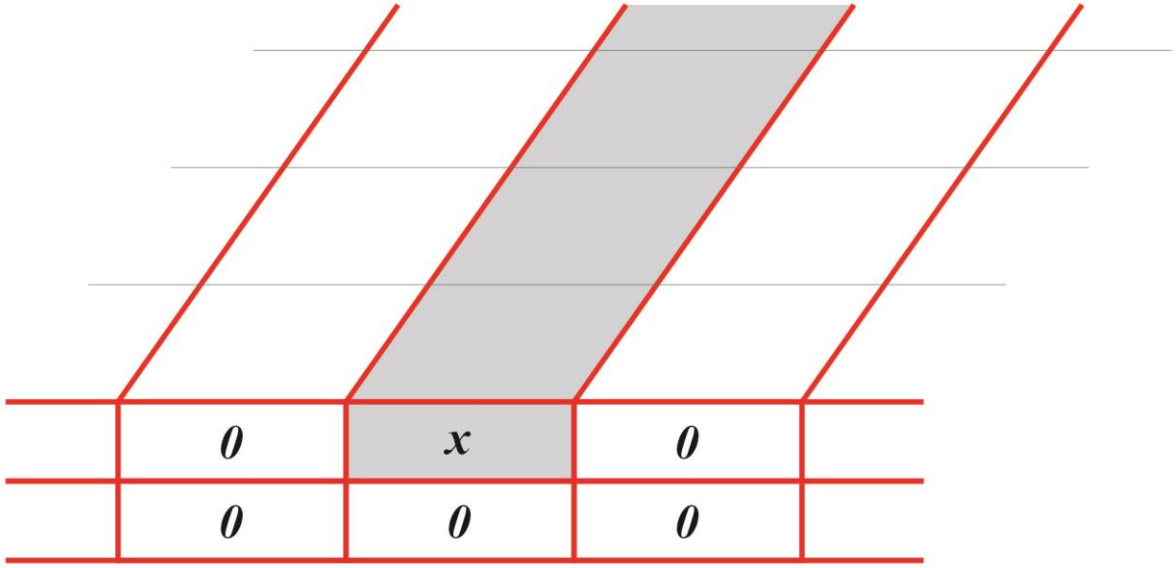
$$\begin{cases} x'_i = F(x_i) \oplus y_i \\ y'_i = x_{i+1} \end{cases}$$

Here $\oplus$ denotes the cell-wise modulo two sum of the two strips.

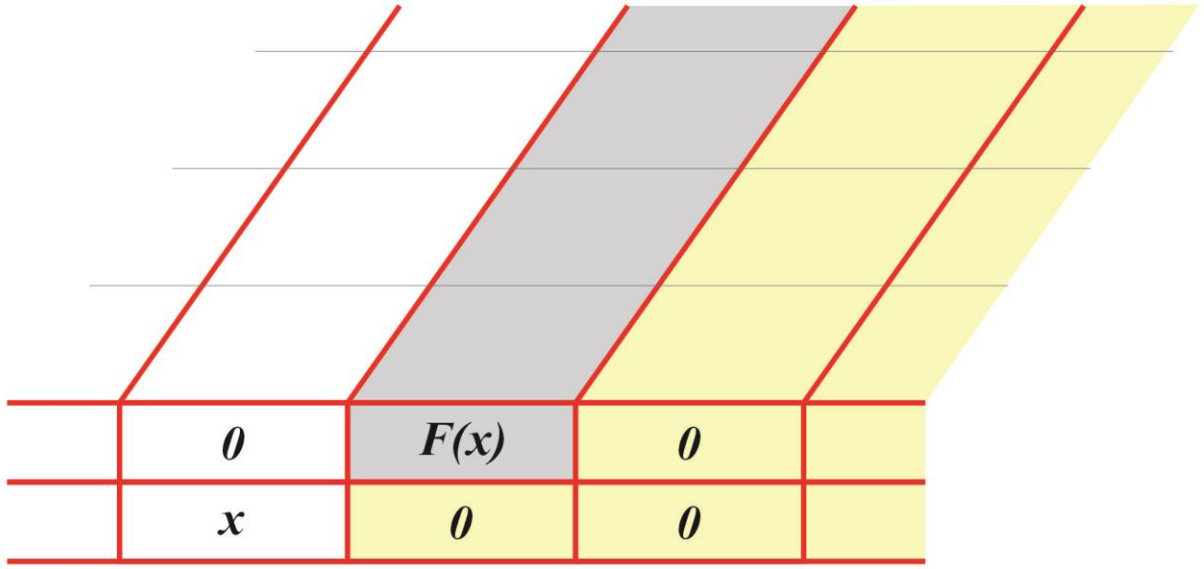


The rule is reversible: old value x_i can be directly read from the lower layer, and then $y_i = F(x_i) \oplus x'_i$:

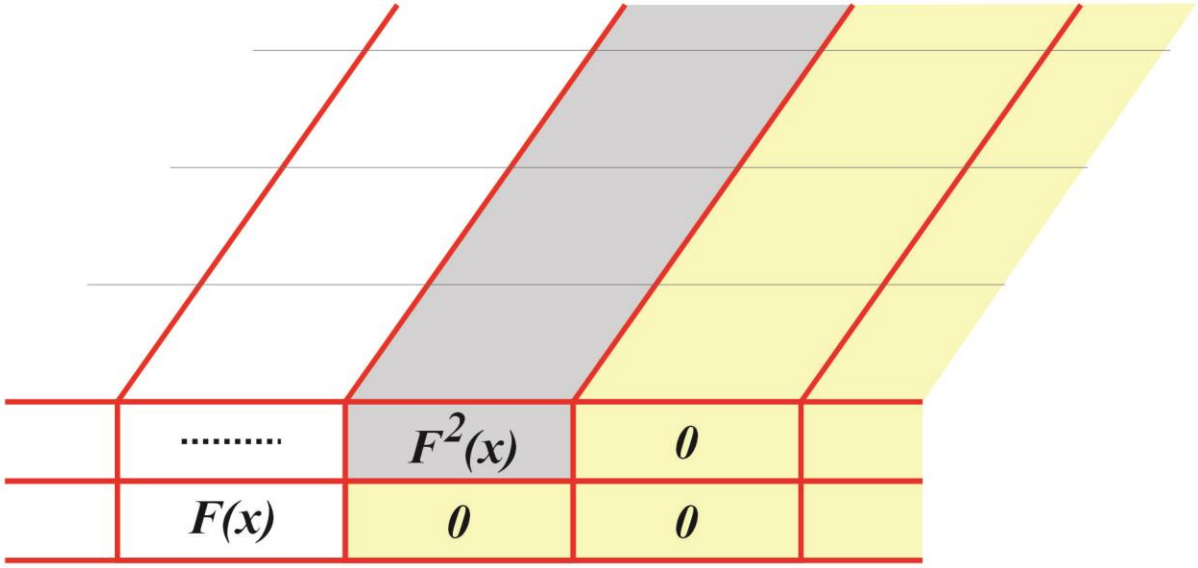
$$\begin{cases} x_i = y'_{i-1} \\ y_i = F(x_i) \oplus x'_i \end{cases}$$



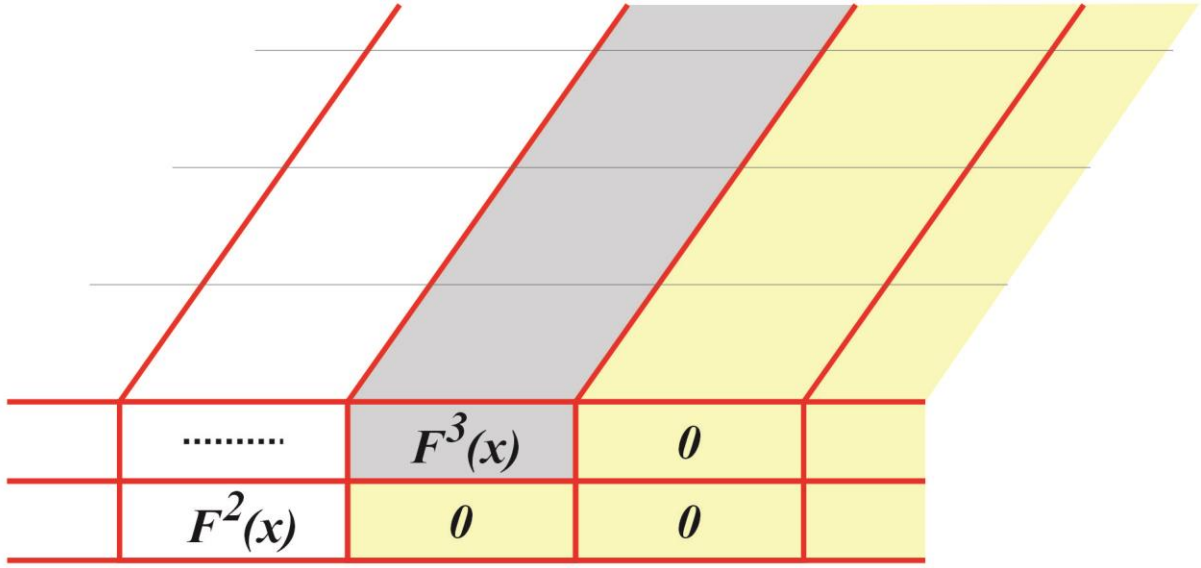
Now the one-dimensional CA *F* gets simulated if the initial configuration is placed into a strip on the upper track, surrounded by 0's everywhere.



- Because *F(…0…)=…0…* and *0* ⊕ *0=0*, the strips on the right remain in state *0* for ever.
- The initial strip thus evolves according to *F*.
- The cells on the left receive history information that keeps on spreading further left.



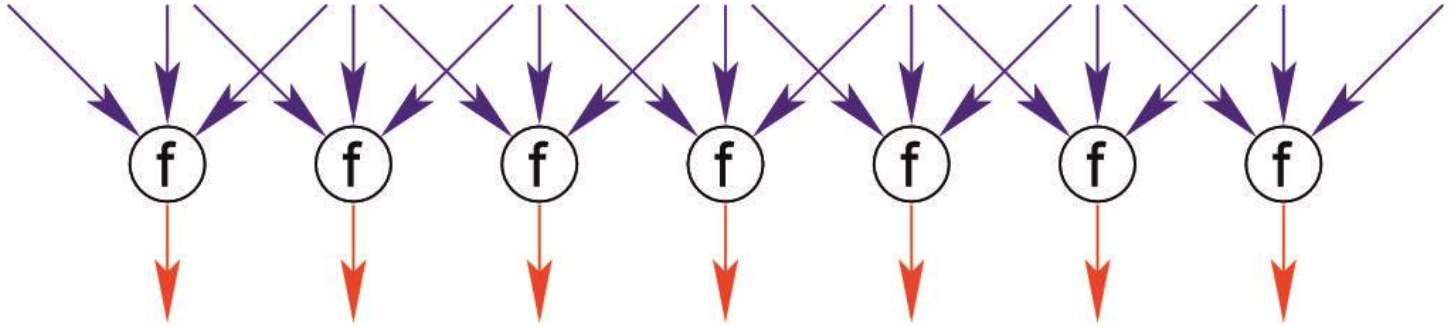
- Because *F(…0…)=…0…* and *0* ⊕ *0=0*, the strips on the right remain in state *0* for ever.
- The initial strip thus evolves according to *F*.
- The cells on the left receive history information that keeps on spreading further left.



- Because *F(…0…)=…0…* and *0* ⊕ *0=0*, the strips on the right remain in state *0* for ever.
- The initial strip thus evolves according to *F*.
- The cells on the left receive history information that keeps on spreading further left.

Toffoli’s 1977 construction left open the one-dimensional case: what is the computational power of one-dimensional RCA ?

Note that reversibility is not directly apparent from the local rule of a cellular automaton. This complicates constructing RCA.



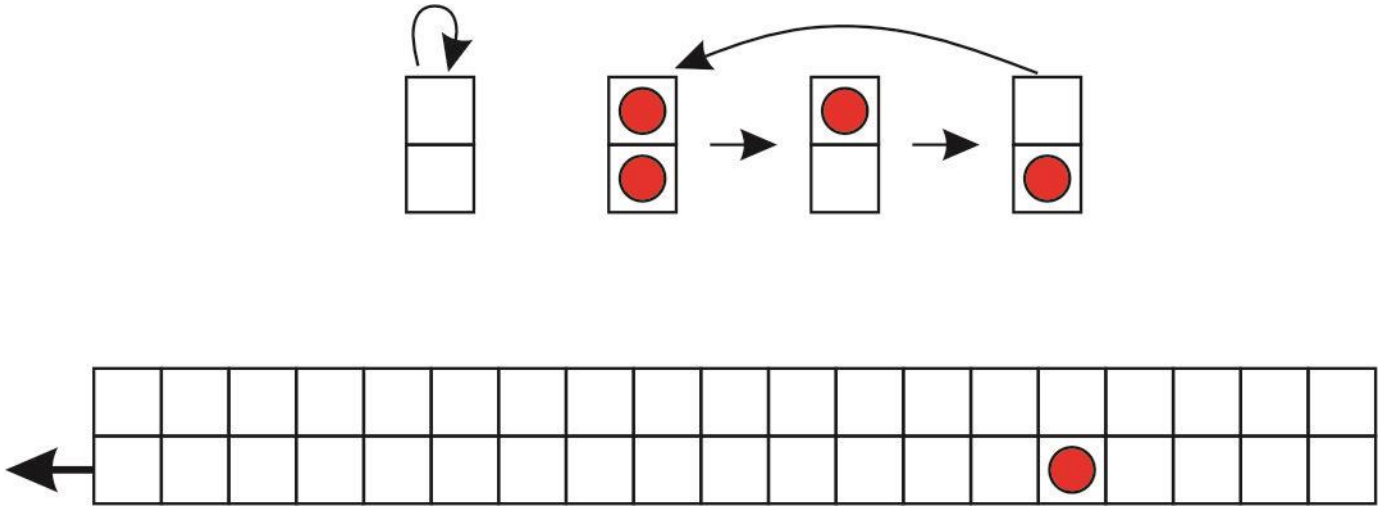
An algorithm was known to determine if a given local rule yields a reversible one-dimensional CA (S.Amoroso, Y.N.Patt 1972) but this hardly helped.

Morita and Harao had the great idea to only consider particular types of cellular automata where reversibility is apparent. These are known as partitioned cellular automata (PCA).

Configurations of PCA form multiple layers. The update is in two stages:

1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

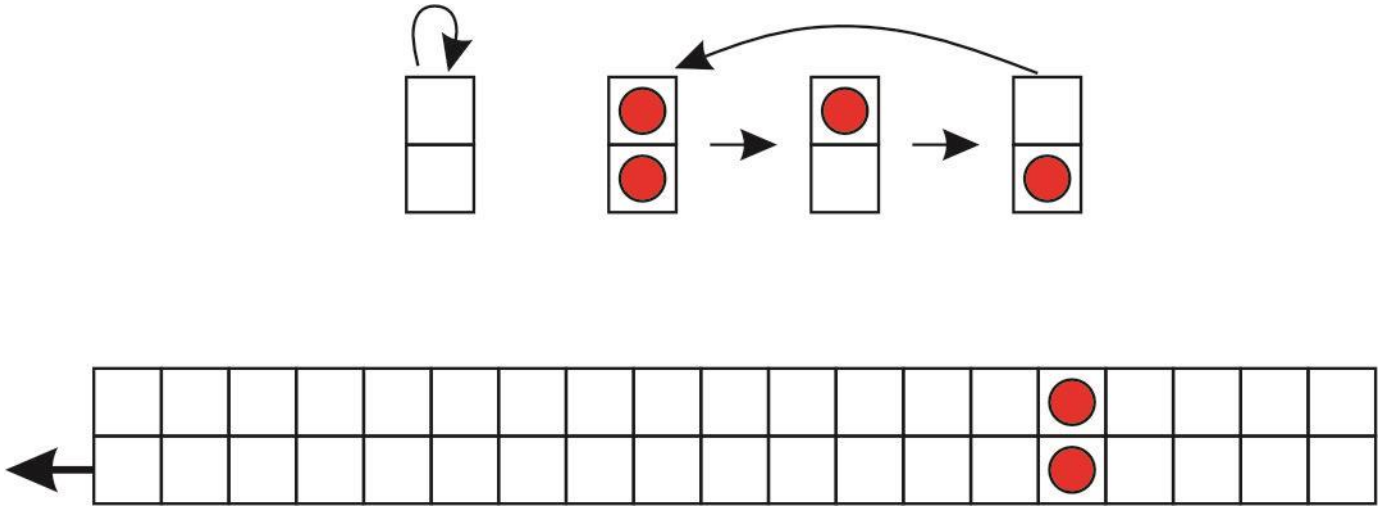
The stages are done alternatingly to evolve the system.



Configurations of PCA form multiple layers. The update is in two stages:

1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

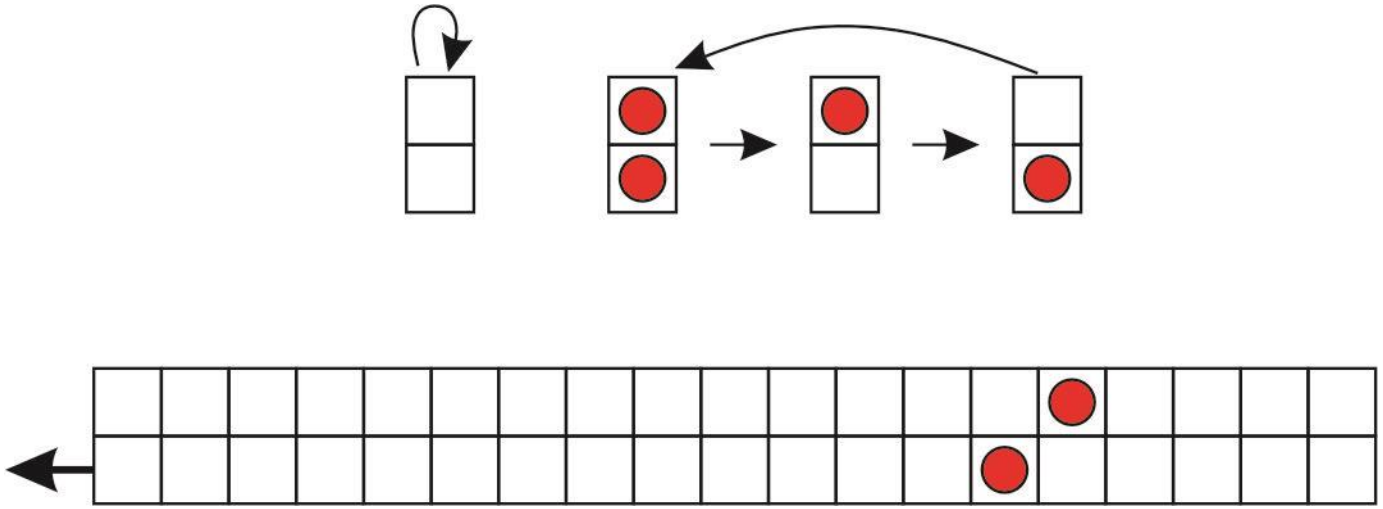
The stages are done alternatingly to evolve the system.



Configurations of PCA form multiple layers. The update is in two stages:

1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

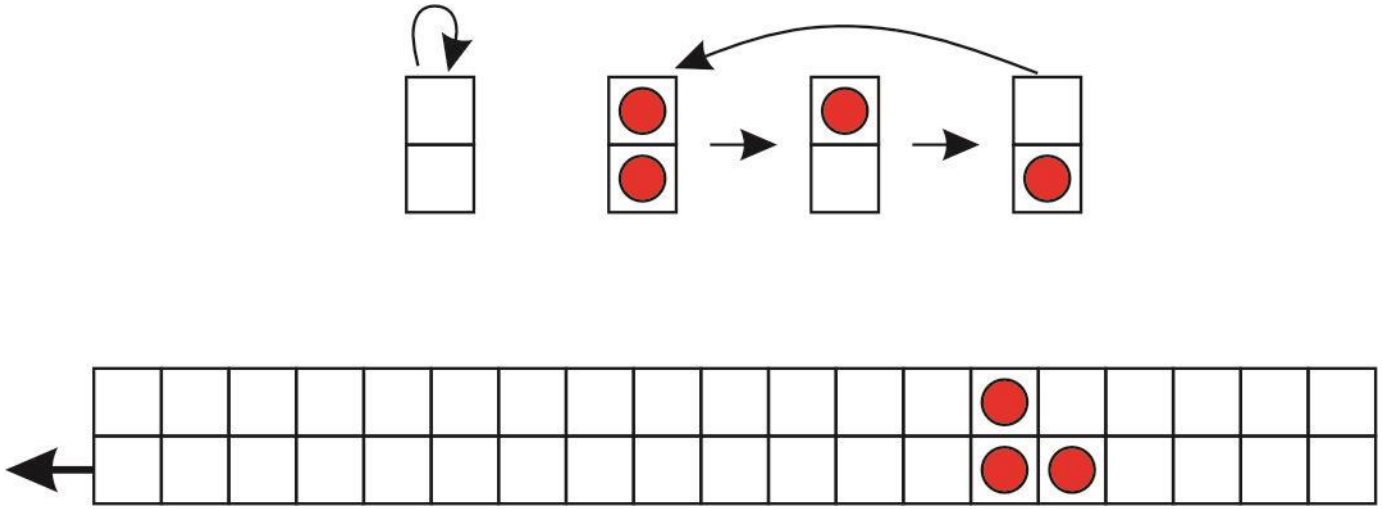
The stages are done alternatingly to evolve the system.



Configurations of PCA form multiple layers. The update is in two stages:

1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

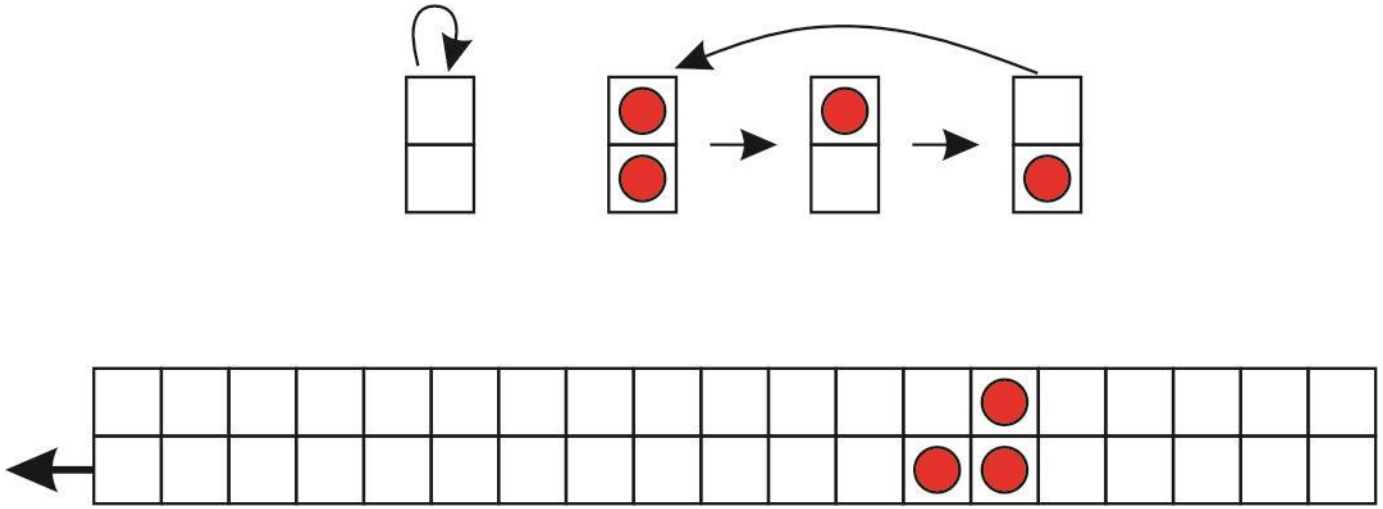
The stages are done alternatingly to evolve the system.



Configurations of PCA form multiple layers. The update is in two stages:

1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

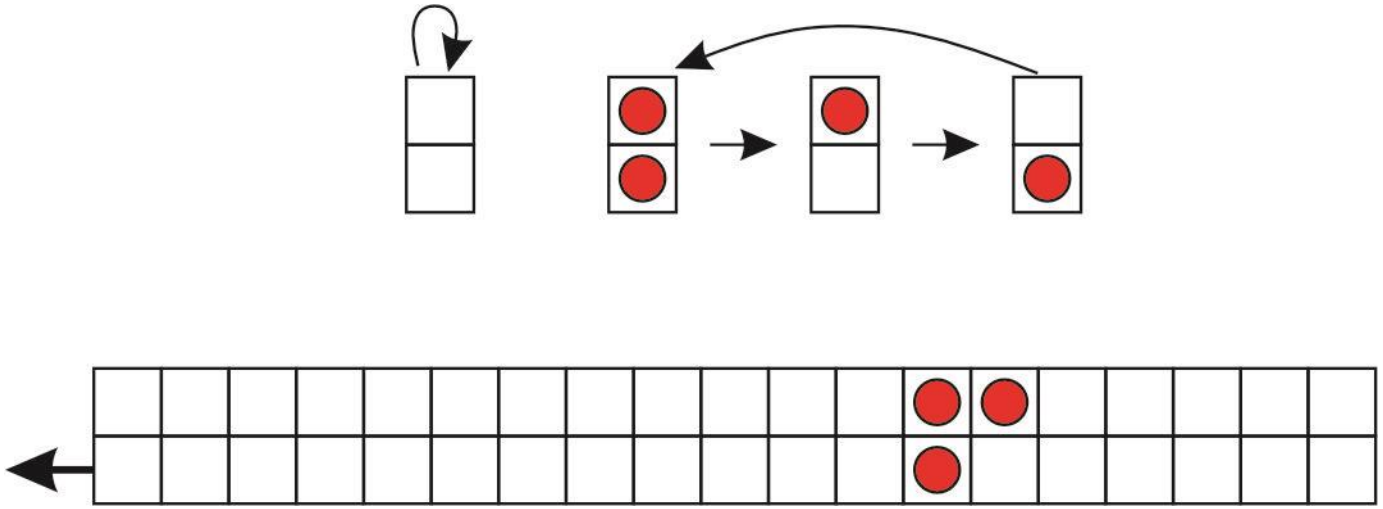
The stages are done alternatingly to evolve the system.



Configurations of PCA form multiple layers. The update is in two stages:

1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

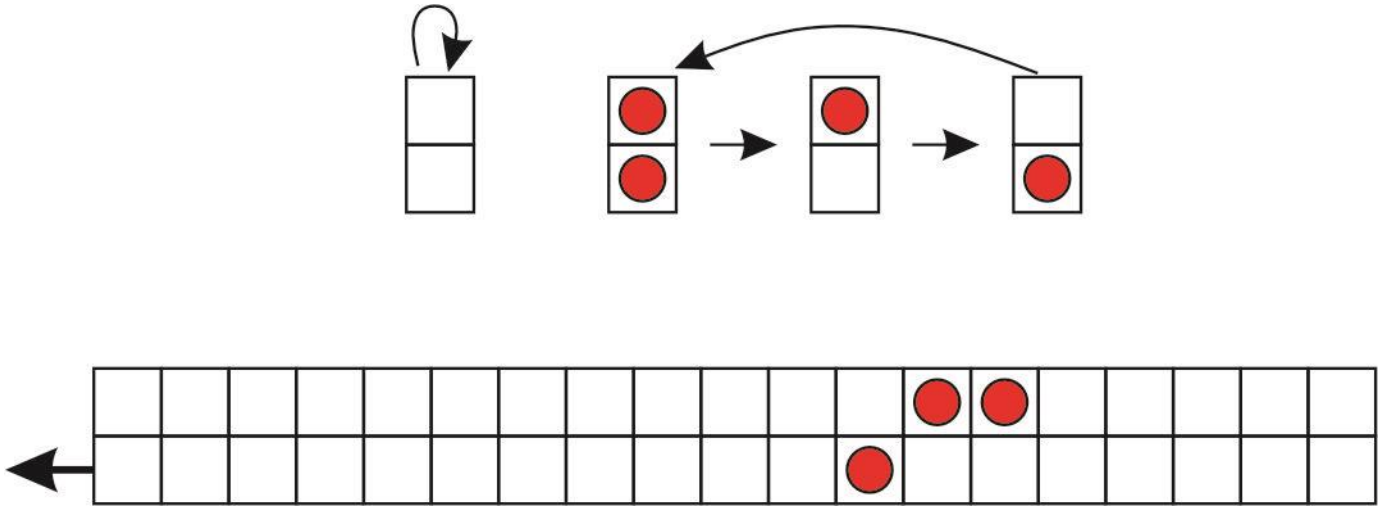
The stages are done alternatingly to evolve the system.



Configurations of PCA form multiple layers. The update is in two stages:

1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

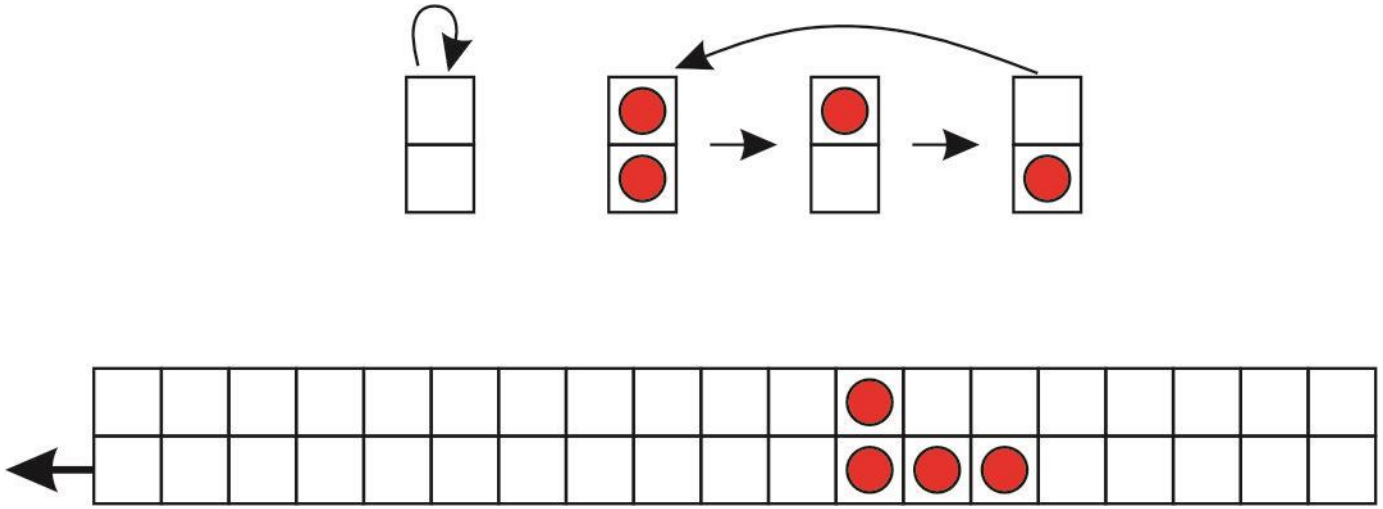
The stages are done alternatingly to evolve the system.



Configurations of PCA form multiple layers. The update is in two stages:

1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

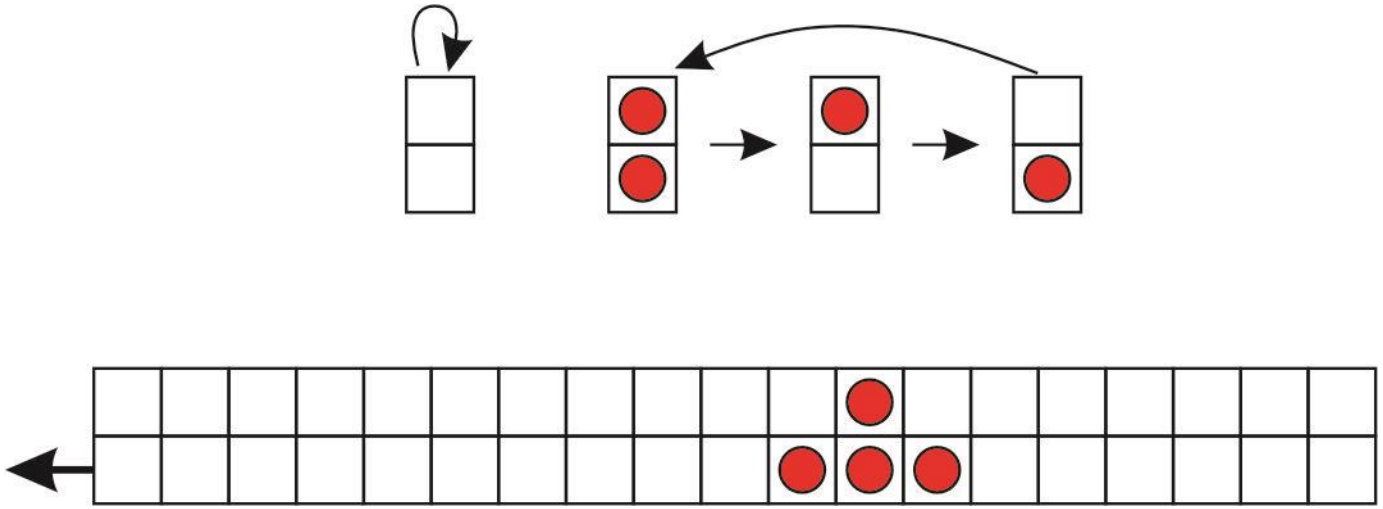
The stages are done alternatingly to evolve the system.



Configurations of PCA form multiple layers. The update is in two stages:

1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

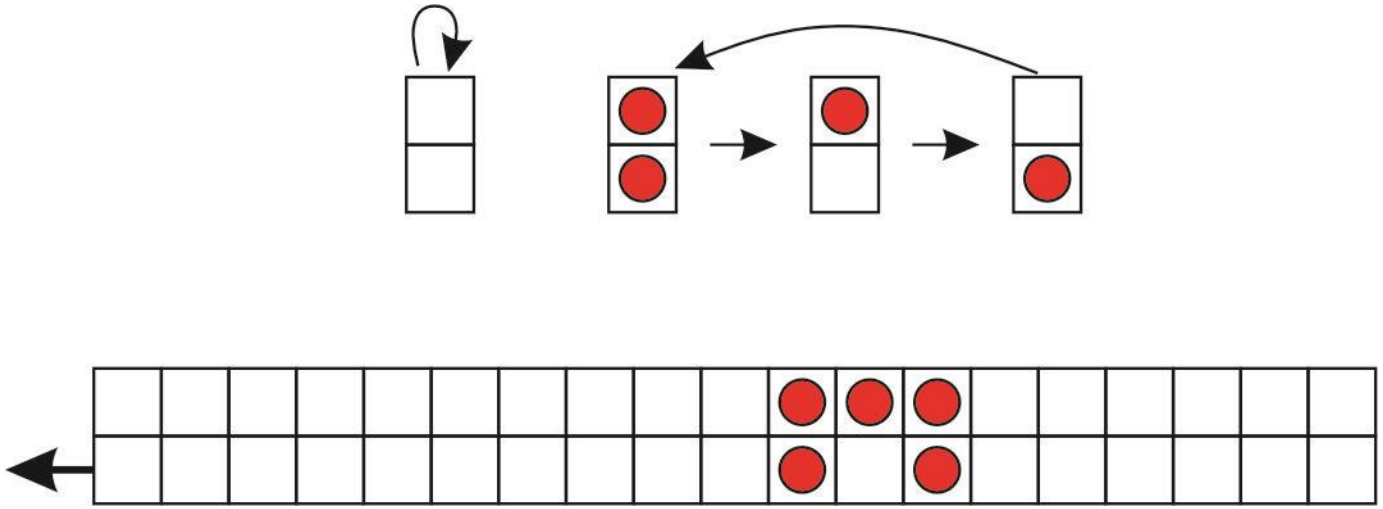
The stages are done alternatingly to evolve the system.



Configurations of PCA form multiple layers. The update is in two stages:

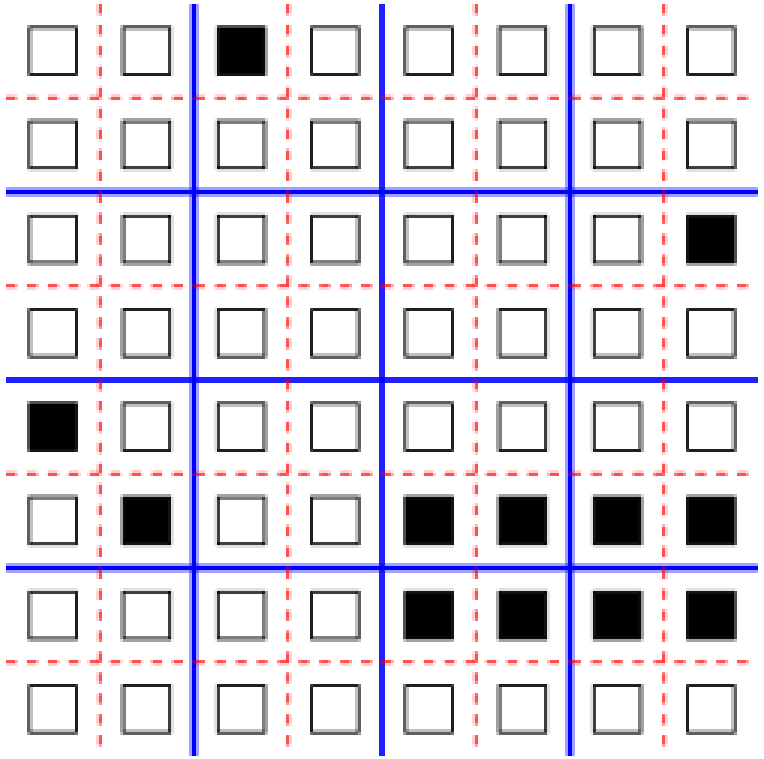
1. A bijection (local rule) of the state set applied at each cell independently of each other.
2. Translations are applied to layers independently of each other.

The stages are done alternatingly to evolve the system.



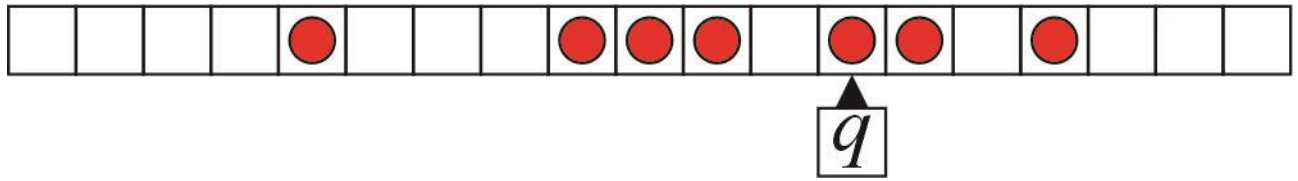
Partitioning is a great way to guarantee reversibility! Partitioning works in the same way in any number of dimensions.

The technique is similar to the earlier technique by Norman Margolus where the grid is partitioned, and a bijection is applied in each part. On alternate steps a different paritioning is used, thus admitting information exhange between cells.



Morita and Harao proved that any reversible Turing machine (RTM) can be embedded in a one-dimensional PCA.

Recall that a Turing machine (TM) consists of an infinite tape and a finite state control unit that accesses one cell of the tape. The control unit has instructions on how to update its state and the tape symbol it is accessing. An instruction may also tell the machine to move on the tape one cell position to the left or to the right.



Morita and Harao use in their paper the quadruple formalism of TM instructions. A rewrite instruction

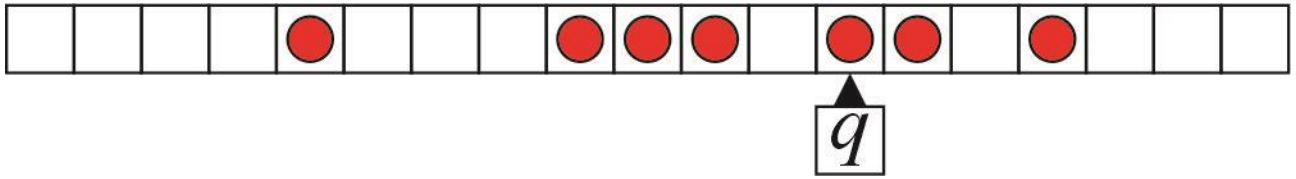
$$(q,a\,|\,p,b)$$

indicates that if the machine is in state q and is scanning tape letter a it may replace a by b and change to state p. No move is done on the tape.

Morita and Harao use in their paper the quadruple formalism of TM instructions. A rewrite instruction

$$(q,a\,|\,p,b)$$

indicates that if the machine is in state q and is scanning tape letter a it may replace a by b and change to state p. No move is done on the tape.

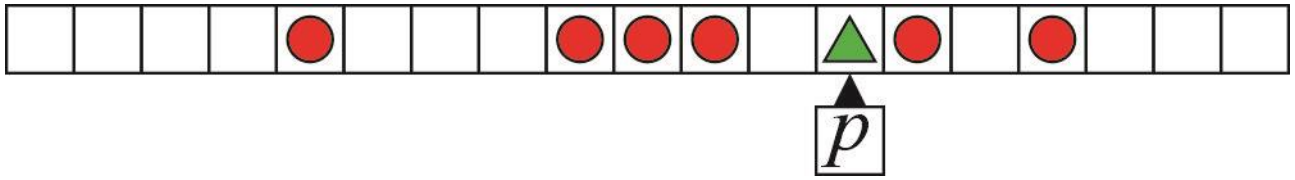


$(q,\bullet\,|\,p,\blacktriangle)$

Morita and Harao use in their paper the quadruple formalism of TM instructions. A rewrite instruction

$$(q,a\,|\,p,b)$$

indicates that if the machine is in state q and is scanning tape letter a it may replace a by b and change to state p. No move is done on the tape.



$(q,\bullet\,|\,p,\blacktriangle)$

A right move instruction

$$(q, p, \rightarrow)$$

indicates that a machine in state q may move on the tape one cell to the right and change its state to p. The left move instruction

$$(q, p, \leftarrow)$$

is defined similarly.

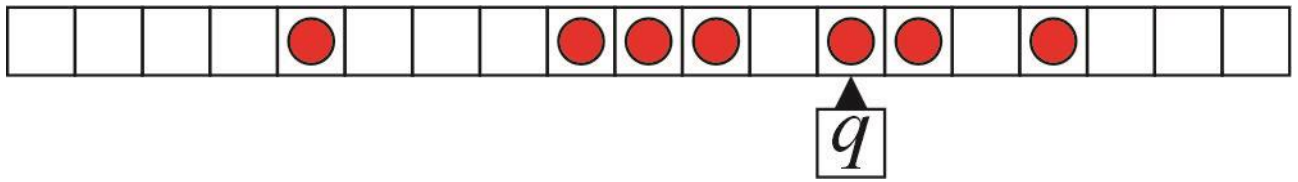
A right move instruction

$$(q, p, \rightarrow)$$

indicates that a machine in state q may move on the tape one cell to the right and change its state to p. The left move instruction

$$(q, p, \leftarrow)$$

is defined similarly.



$$(q, p, \leftarrow)$$

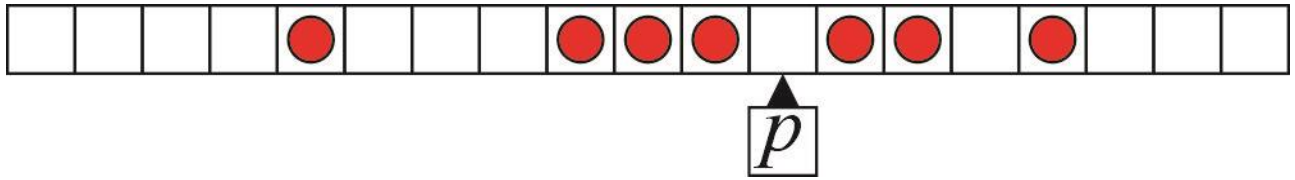
A right move instruction

$$(q, p, \rightarrow)$$

indicates that a machine in state q may move on the tape one cell to the right and change its state to p. The left move instruction

$$(q, p, \leftarrow)$$

is defined similarly.



$$(q, p, \leftarrow)$$

A set of instructions defines a TM program.

In a non-deterministic TM there may be several instructions applicable simultaneously.

In a deterministic TM at most one instruction may be applicable in any situation. This can be easily checked from the TM program:

- For each q and a the program may contain at most one rewrite instruction $(q,a\,|\,*,*)$
- For each q the program may contain at most one move instruction $(q,*,*)$
- If there is a move instruction $(q,*,*)$ from a state q then there may be no rewrite instruction $(q,*\,|\,*,*)$ from the same state q.

The inverse instruction retracks a TM operation:

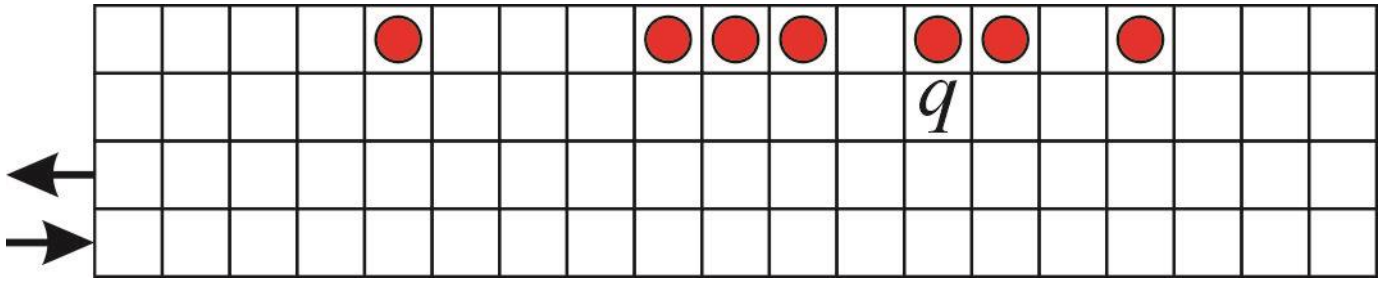
- The inverse of $(q,a|p,b)$ is $(p,b|q,a)$
- The inverse of $(q, p, \rightarrow)$ is $(p, q, \leftarrow)$
- The inverse of $(q, p, \leftarrow)$ is $(p, q, \rightarrow)$

The inverse TM of a machine M is a TM whose program contains precisely the inverse instructions of M. The inverse of M precisely retraces the computations of M back in time.

A TM is reversible (RTM) if it is deterministic and also the inverse machine is deterministic.

For any RTM *M* Morita and Harao constructed a simulating PCA *A* with four tracks:

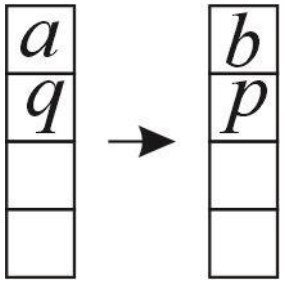
- The first two tracks store the tape content and the control unit of *M*.
- The other two tracks are empty conveyer belts moving left and right. When *M* needs to move left/right the state is placed on the corresponding conveyer belt.

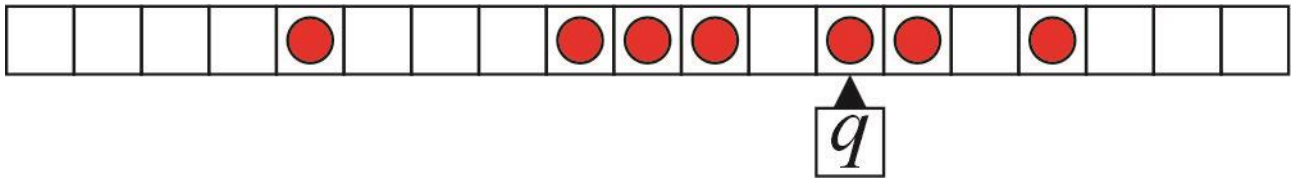


For each rewrite instruction

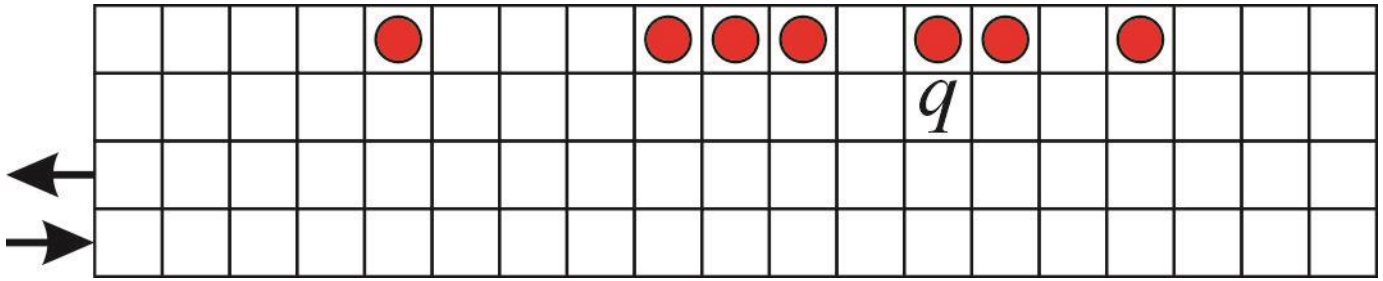
$(q,a|p,b)$

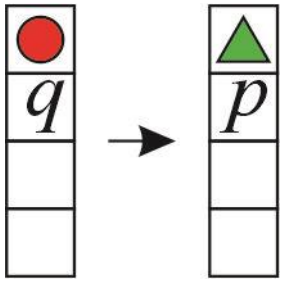
The PCA local rule maps

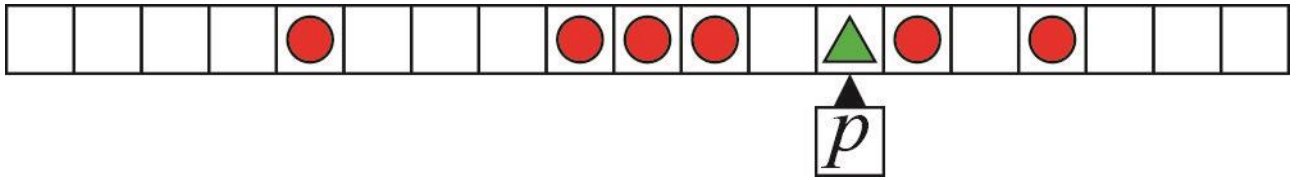




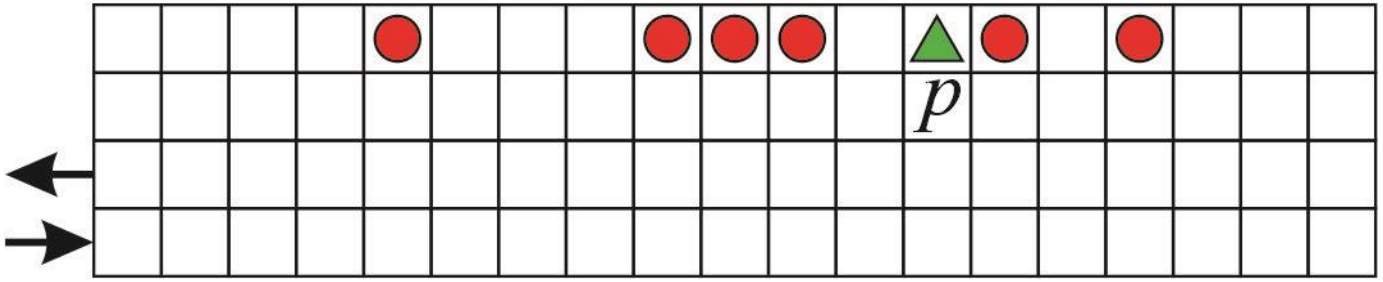
$(q, \bullet \mid p, \blacktriangle)$

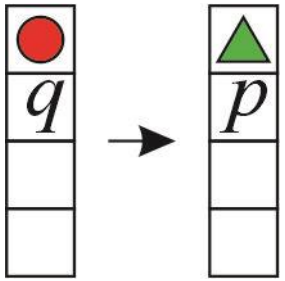






$(q, \bullet \mid p, \blacktriangle)$

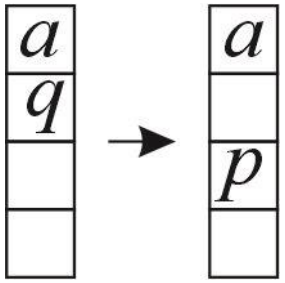




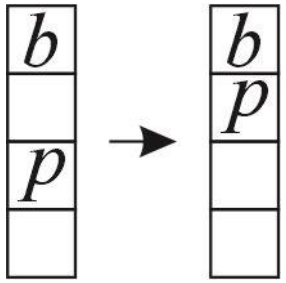
For each left move instruction

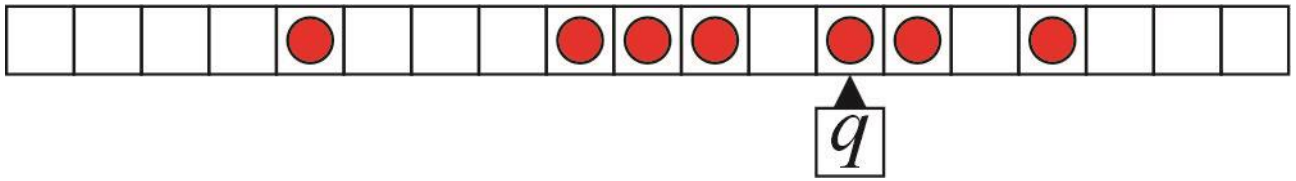
$(q, p, \leftarrow)$

The PCA local rule maps for all a:

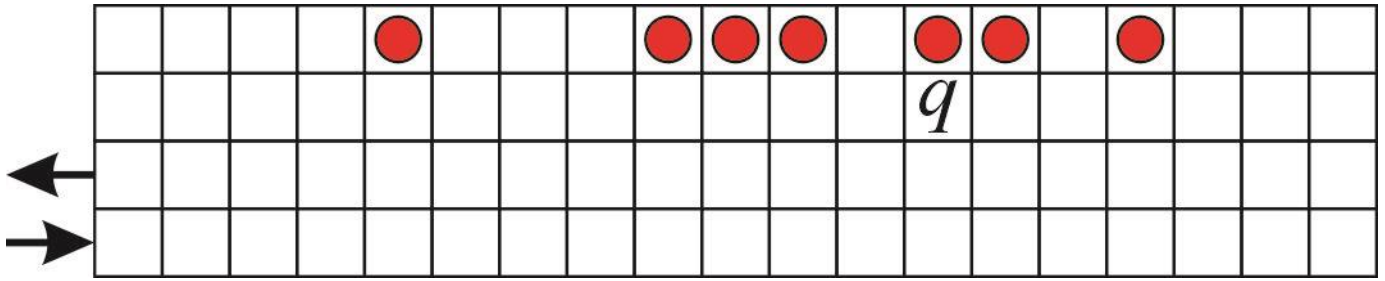


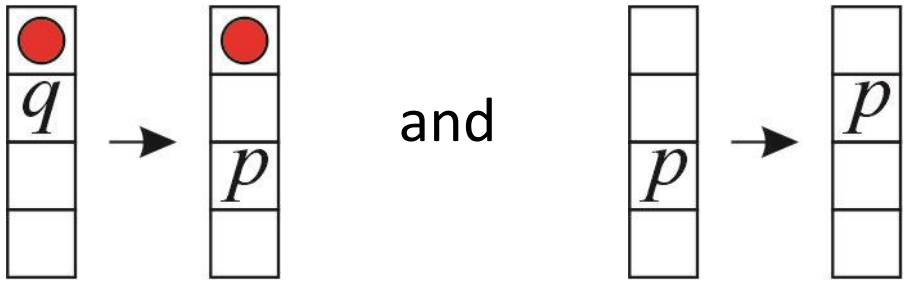
and for all b:



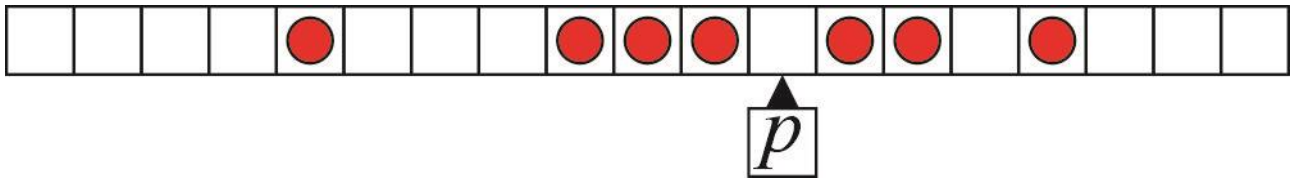


$(q, p, \leftarrow)$

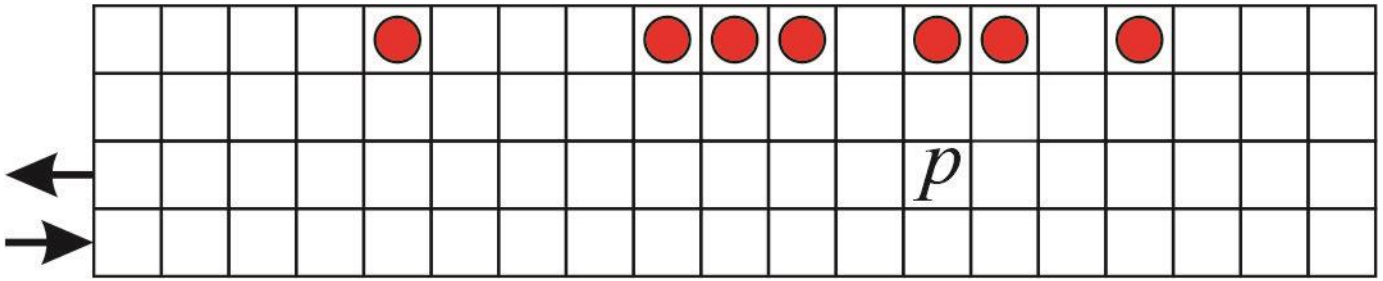


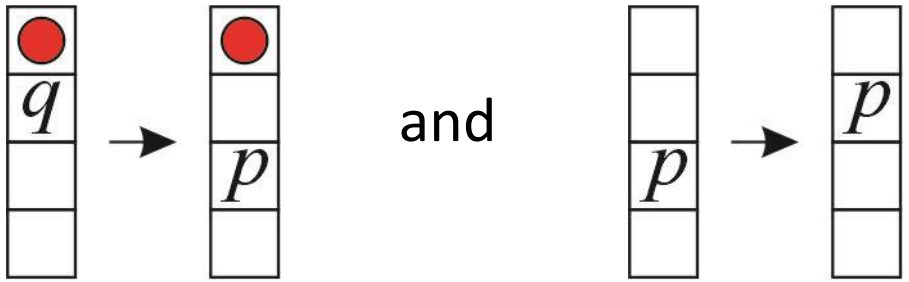


and

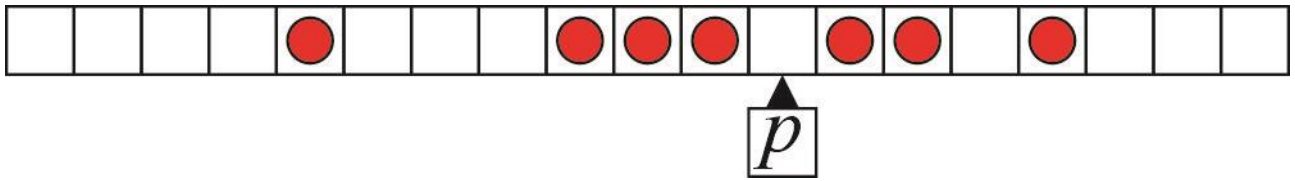


$(q, p, \leftarrow)$

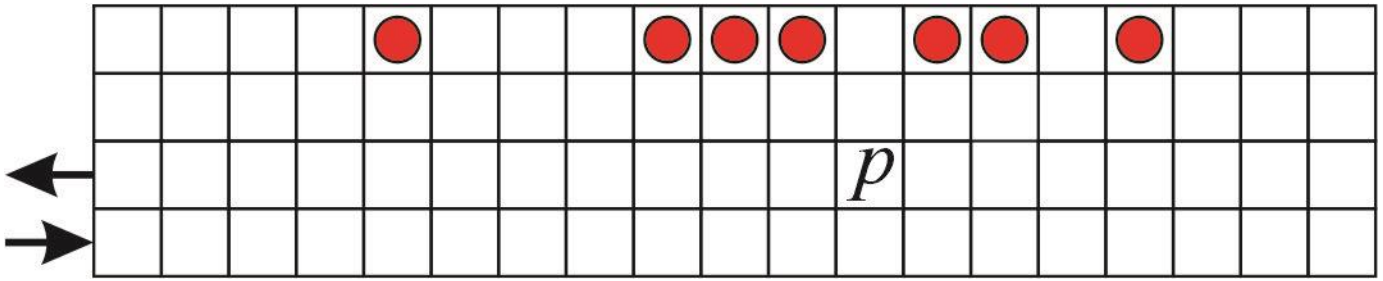


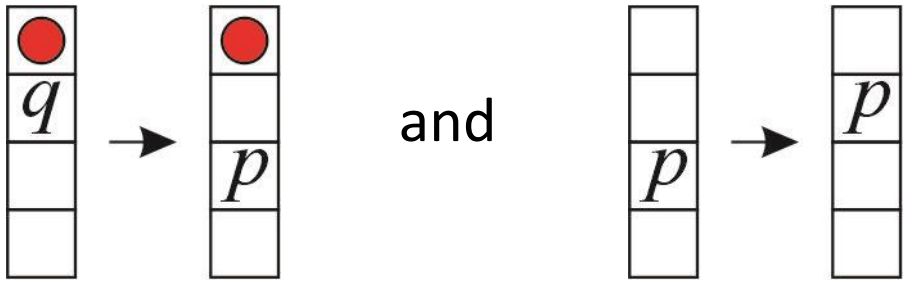


and

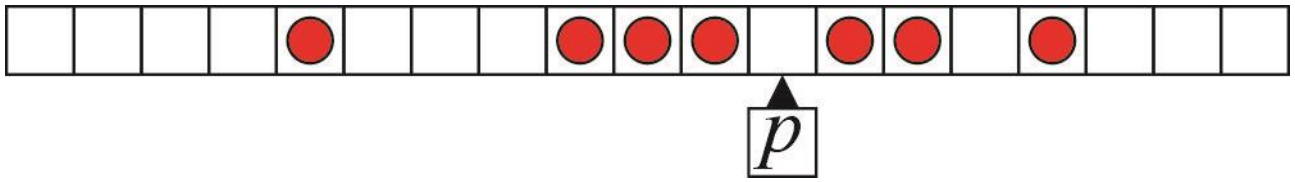


$(q, p, \leftarrow)$

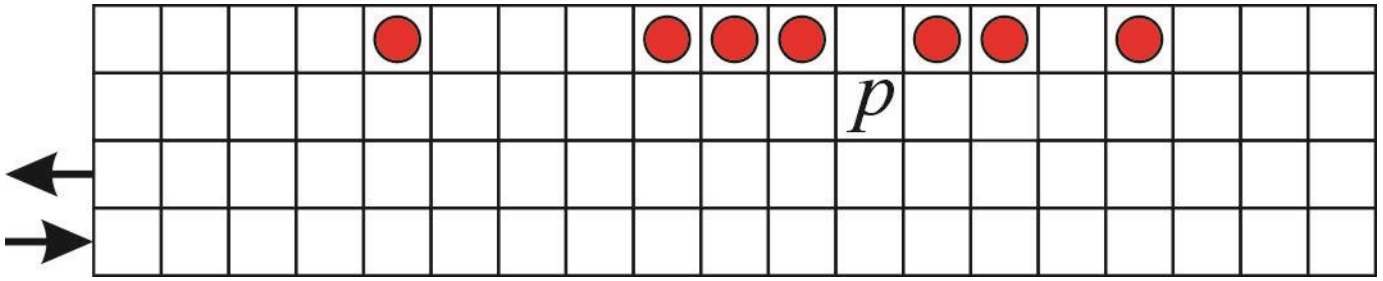


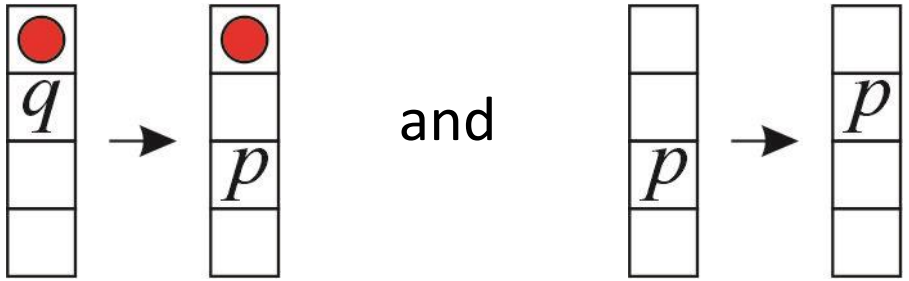


and



$(q, p, \leftarrow)$



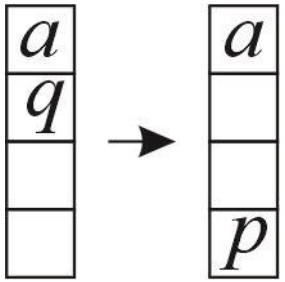


and

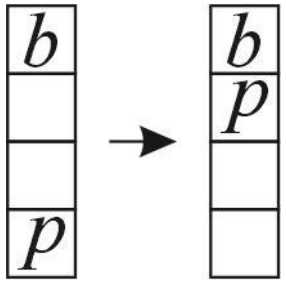
Similarly, for each right move instruction

$(q, p, \rightarrow)$

The PCA local rule maps for all a:

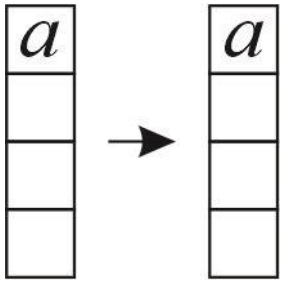


and for all b:



Finally, the cells without the TM control unit should not be changed.

So the PCA local rule maps for all a:



With these local rule values the PCA *A* simulates the RTM *M*: each rewrite instruction of M is simulated in a single step, and each moving instruction in two steps.

But so far the local rule is only partially defined: there are many PCA states that do not occur in any valid RTM simulation, e.g.,

But the reversibility of the Turing machine means that the partially defined local rule is injective. Thus the number of unused PCA states in the range and in the domain of the local rule are identical. We can arbitrarily match these with each other to complete the local rule into a bijection.

But so far the local rule is only partially defined: there are many PCA states that do not occur in any valid RTM simulation, e.g.,

But the reversibility of the Turing machine means that the partially defined local rule is injective. Thus the number of unused PCA states in the range and in the domain of the local rule are identical. We can arbitrarily match these with each other to complete the local rule into a bijection.

This was the Morita-Harao construction.

The partitioning technique to guarantee reversibility opened up new vistas to constructing small universal RCA, especially in higher dimensions.

Already in 1992 Kenichi proposed with S. Ueno universal PCAs in two-dimensions with four binary layers. Two rules were proposed, both state conserving and also rotation invariant.

IEICE TRANS. INF. & SYST., VOL. E75-D, NO. 1 JANUARY 1992

PAPER Special Section on Theoretical Foundations of Computing

Computation-Universal Models of Two-Dimensional 16-State Reversible Cellular Automata

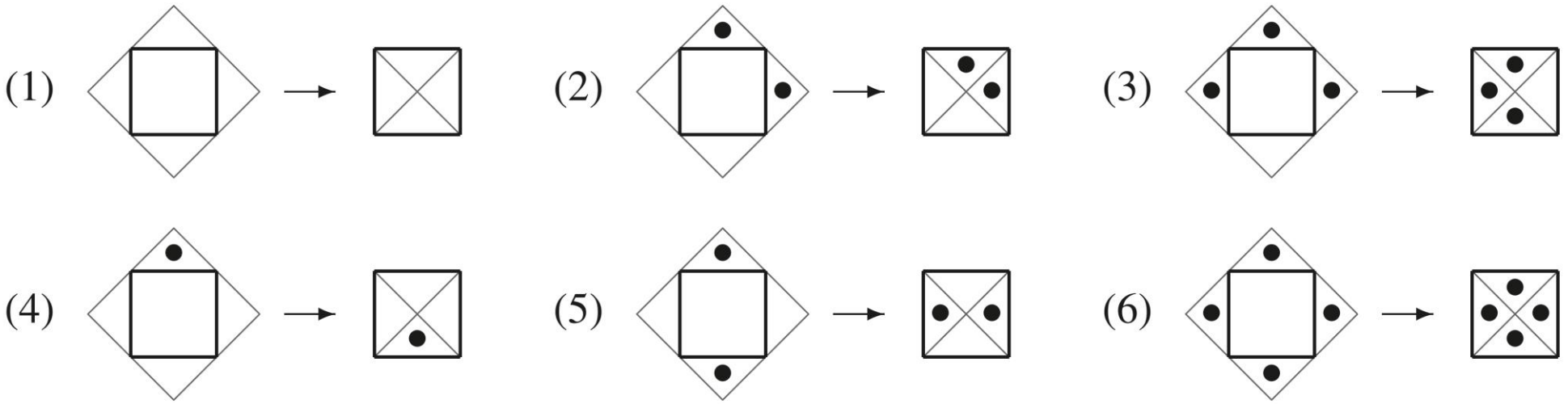
Kenichi MORITA† and Satoshi UENO†, Members

SUMMARY A reversible (or injective) cellular automaton (RCA) is a "backward deterministic" CA, i.e., every

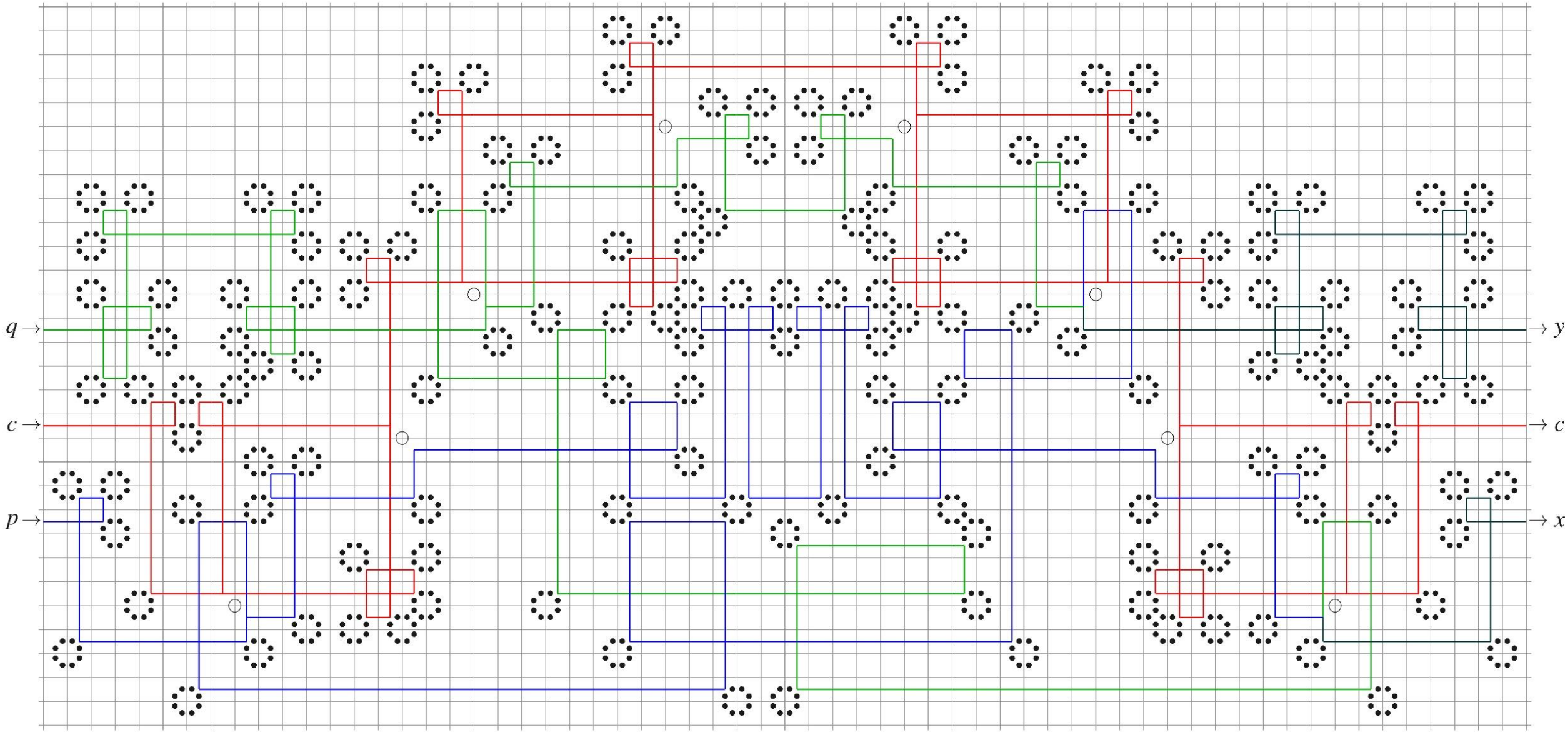
simple universal RCA can be constructed. Though his model is interesting, it differs from the standard CA

The first rule is a PCA reformulation of the billiard-ball CA by Margolus.

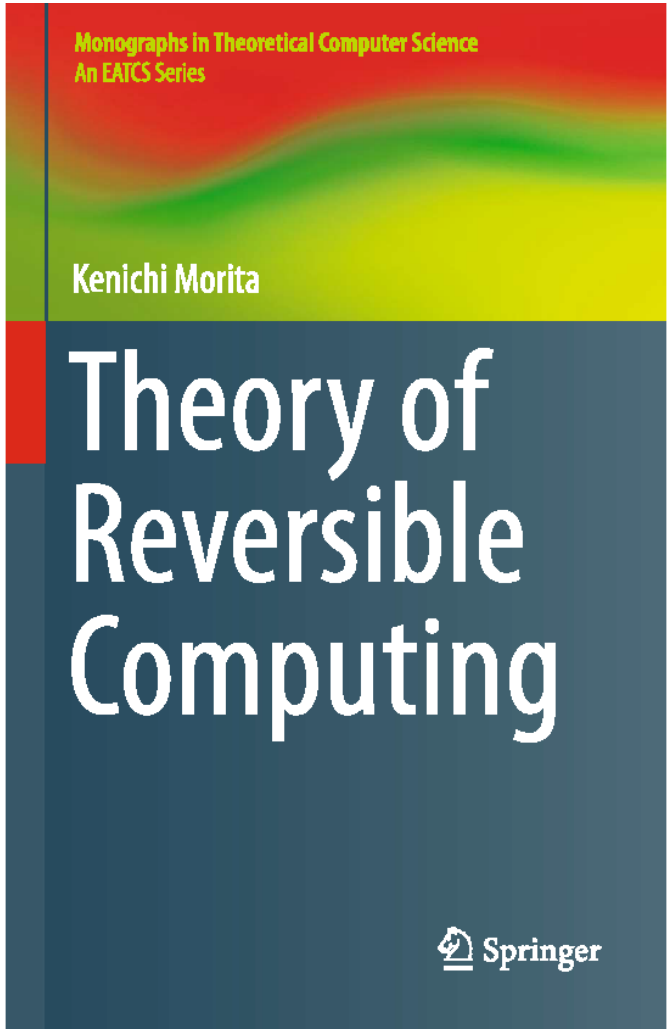
The second one has the following local rule:

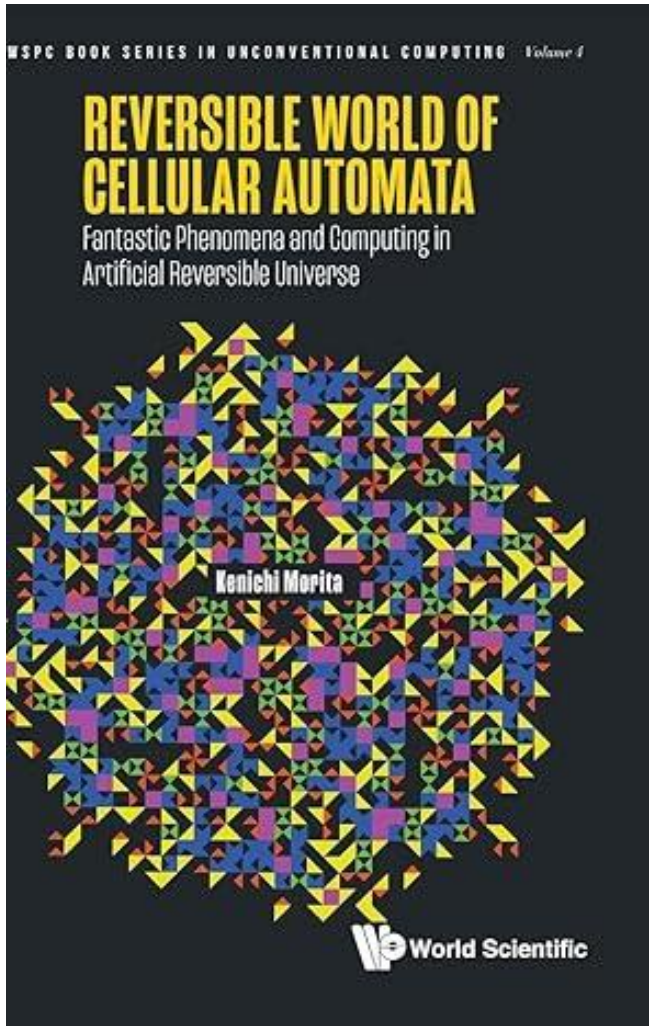


Here's the Fredkin gate implementation using the two-dimensional PCA:

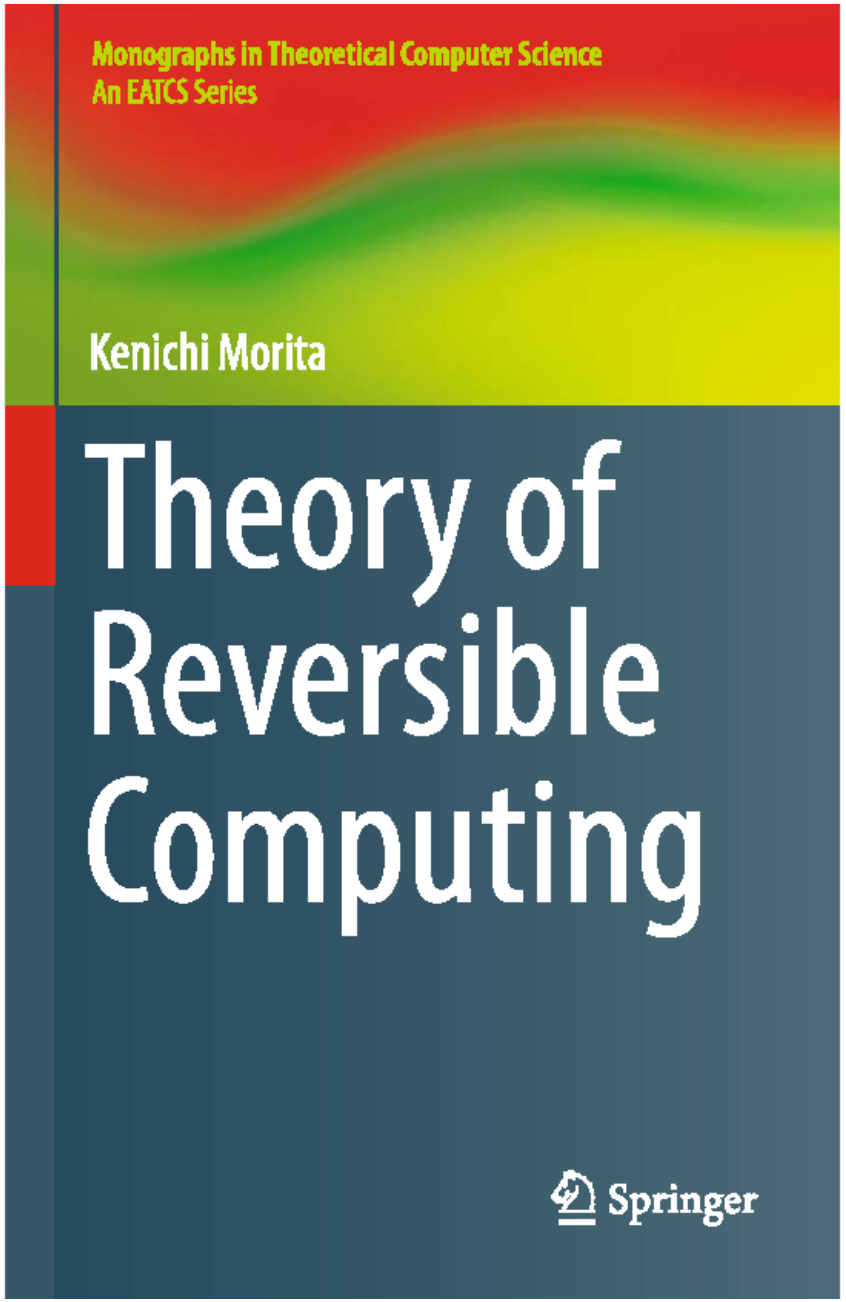


Kenichi was a master of such amazing PCA constructions. His books are a source of inspiration, containing many wonderful PCAs.





Especially Morita's 2017 monograph on theory of reversible computing is a manifestation of his central role in developing the theory of reversible computation, and in designing small/simple computationally universal reversible devices.

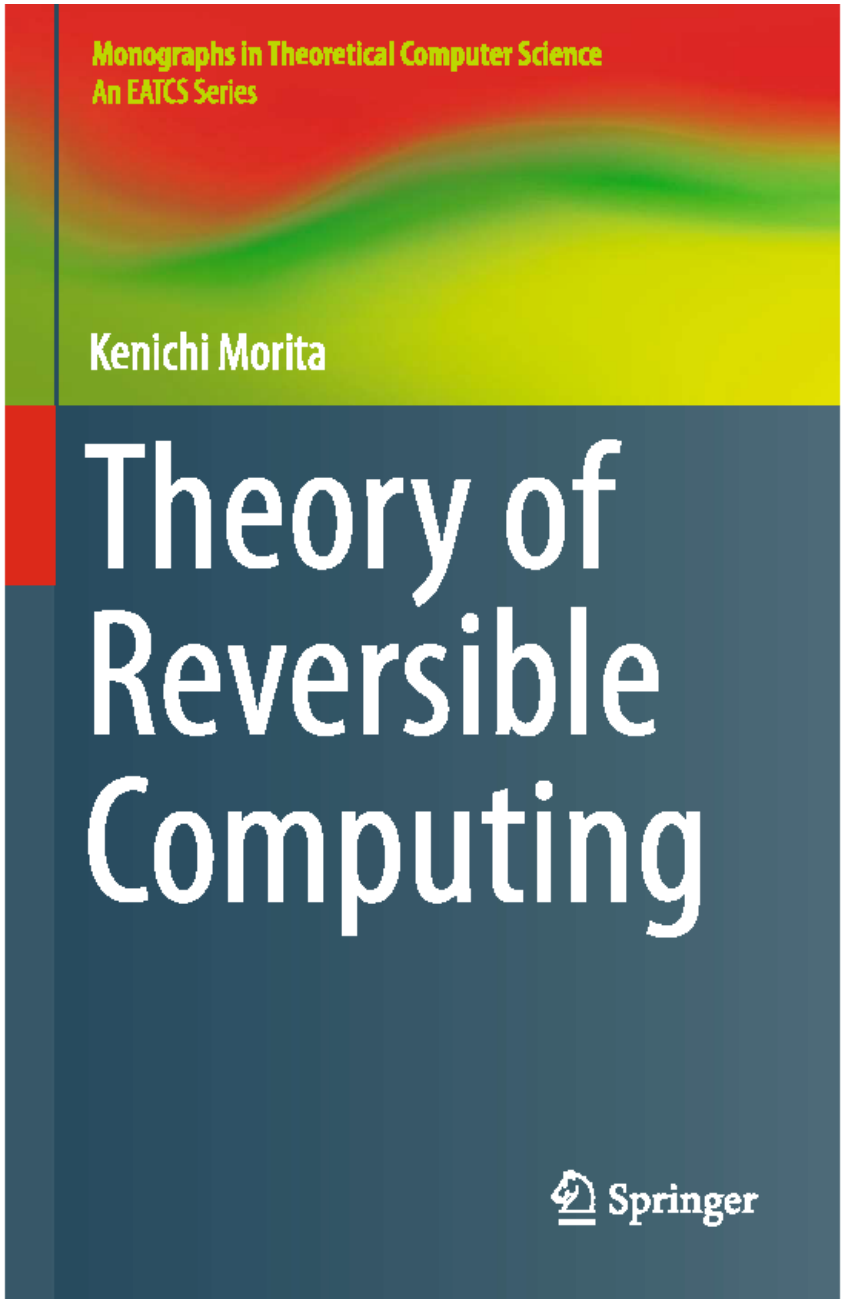


Especially Morita's 2017 monograph on theory of reversible computing is a manifestation of his central role in developing the theory of reversible computation, and in designing small/simple computationally universal reversible devices.

Here's a quote from the Preface of the book:

This book is not a comprehensive textbook on reversible computing, but describes mainly the results shown in the papers by myself and my colleagues, which were published between 1989 and 2017.

Few people can say this about a book of 450 pages!



In this talk I reviewed Kenichi Morita's classical paper from 1989.

This paper started Kenichi's amazing path in reverible cellular automata constructions. His ideas will continue to provide insipiration to us and many future generations of scientists.

For example, just this month Matthew Cook and Ethan Palmiere presented at UCNC the paper

The Morita Gate is Universal

where they show that one of the 24 possible two input bit, binary state RLEMs (reversible logical element with memory) is universal. This was left as an open problem in Kenichi's paper in 2012.

Cook and Palmiere named this particular RLEM the Morita gate to honor the tremendous contributions by Kenichi in the investigations on computational universality of RLEMs.