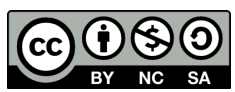
UCT

Reversible Computation

Klaus Sutner

Carnegie Mellon University

Spring 2024



Fugaku

442 peta flops, 7630848 cores, cost \$1.2 billion, power consumption 29.9 MW.

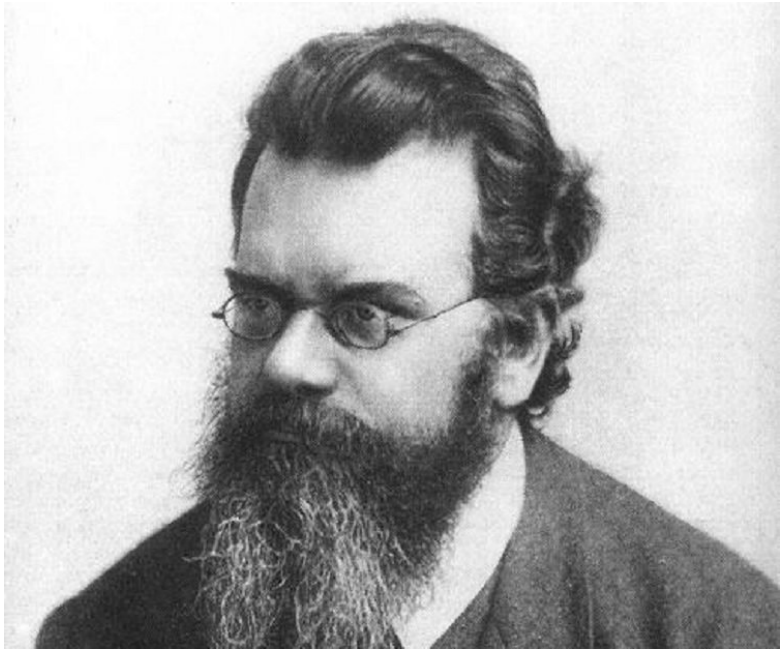
In the 1950's John von Neumann speculated that the energy cost associated with manipulating a single bit is at least

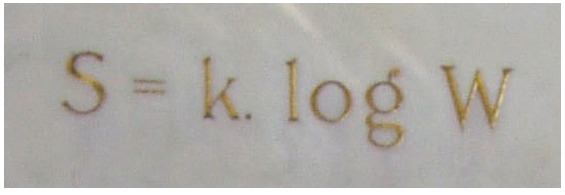
$$k\,T\,\ln 2.$$

Here T is the absolute temperature, and k is Boltzmann's constant:

$$k \approx 1.3806503 \times 10^{-23}\,JK^{-1}$$

It defines the relation between absolute temperature and the kinetic energy contained in each molecule of an ideal gas.





Statistical mechanics and atomic theory in the late 19th century. Heat is just the kinetic energy of moving atoms/molecules (kinetic molecular theory of heat).

Highly contested at the time.

Around 1960 Rolf Landauer took a closer look at the problem and found the following result, a first hard lower bound on the physical cost of computation.

Theorem (Landauer 1961)

Erasing a single bit requires at least $kT \ln 2$ *energy.*

It is important to understand that this is not a question of technology, it's a question of thermodynamics, of fundamental physics.

There is no way around this fundamental cost of erasure – unless our understanding of physics is all wrong.

Current computers are much, much worse: they dissipate energy even when nothing is happening (flip-flops) as far as the computation is concerned.

Typically a real PC spends 10^{12} times as much energy as the thermodynamical limit.

Biological computing is slightly better, neurons seem to dissipate about $10^{11} kT$. Darwin still beats Intel by a factor of 10.

But DNA computing is much superior; it seems to use only about 100 times the Landauer limit. That's 10 orders of magnitude better than a PC. Of course, DNA computing is utterly impractical.

As everyone knows, time flows in one direction and one direction only: if an egg has dropped to the floor and is broken, there is no magic switch to flip that will cause the egg to re-assemble. According to everyday observation, there is a pronounced asymmetry in time.

Yet, at the basic level the laws of physics seem to be reversible with respect to time, there is no fundamental reason why some physical process could not run backwards, the equations that describe the process are perfectly symmetric.

In particular the Second Law of Thermodynamics which describes macroscopic systems (such as a container filled with bouncing gas molecules) appears to collide head-on with time-symmetry.

This conundrum is known as Loschmidt's paradox.

The second law and time symmetry are both well-established principles in physics, with good observational and theoretical support. And yet they clash.

Boltzmann is supposed to have responded to his archenemy Loschmidt's criticism like so:

> Go ahead, reverse them!

Landauer showed the following amazing result that establishes a connection between logic and physics.

Theorem

The only computer operations that must be thermodynamically irreversible are the logically irreversible ones.

This is utterly insane. What does computation have to do with physics? How can a purely logical concept relate to the laws of physics?

At any rate, Landauer's result suggests that could avoid any thermodynamic penalty as long as we use a reversible computer. We have to pay for erasing bits, but not necessarily for anything else.

Several questions come to mind.

- What exactly is a logically reversible operation?
- Can all computations be carried out in a reversible manner?
- Is there any chance to build a reversible computer?

What exactly is this supposed to mean?

In any model of computation, we have some space of configurations $\mathcal{C}$ and a next-step relation $\rightarrow$, together with special initial configurations C_x^{init} and final configurations C_y^{halt}. Think of $\mathfrak{C} = \langle \mathcal{C}, \rightarrow \rangle$ as a directed graph.

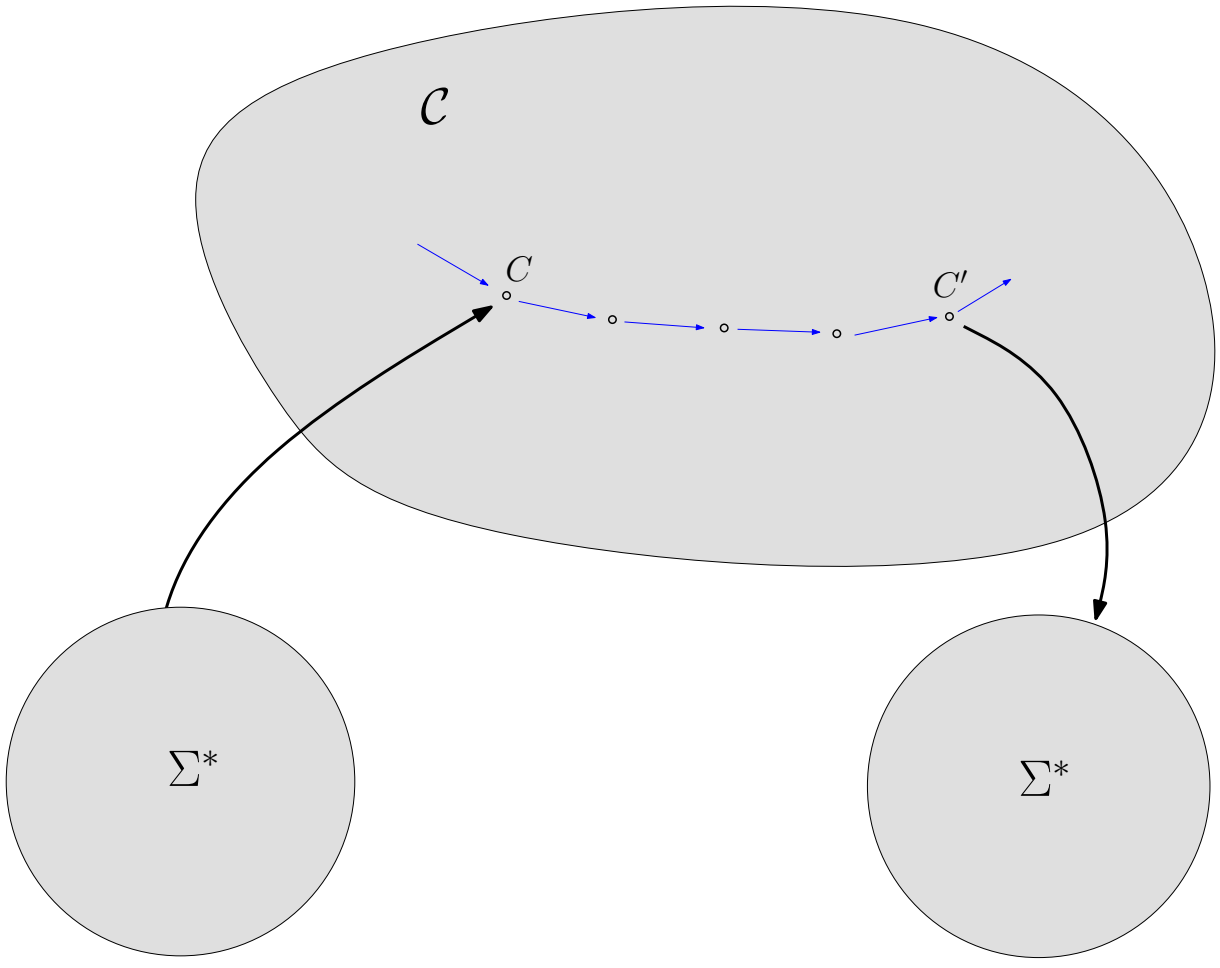
We are only interested in deterministic models, so all nodes have out-degree at most 1. For reversibility we also need in-degree at most 1:

$$C_0 \rightarrow C, C_1 \rightarrow C \qquad \text{implies} \qquad C_0 = C_1$$

So we can follow computations backwards from C_y^{halt} to C_x^{init}.

This is highly unusual, just about any model of computation you can think of is naturally non-reversible.

Compute Space



Disjoint Computations

Consider the computation on input x, a path in $\mathfrak{C}$:

$$C_x^{\mathsf{init}} = C_0, C_1, C_2, \ldots, C_{t-1}, C_t = C_y^{\mathsf{halt}}$$

For reversibility, none of these computations can have any nodes in common, the whole computation graph is just a disjoint union of such overlap-free paths.

Our definition is slightly off in that the only interesting subgraph of $\mathfrak{C} = \langle \mathcal{C}, \to \rangle$ is the part that is reachable from the initial configurations C_x^{init}.

The remainder of the graph cannot appear in any computation, it does not matter what the degrees there look like.

Exercise

Show that in the Turing machine case it is undecidable whether a configuration is in the reachable part.

The no-overlap condition causes a major problem: suppose we wish to compute a function

$$f : \mathbf{2}^\star \to \mathbf{2}^\star$$

reversibly. Then f must be injective, otherwise the paths of $f(x) = f(x')$ where $x \neq x'$ would overlap, at at least in the last configuration.

To accommodate other functions (just think about addition or multiplication of natural numbers), we consider the inflated version

$$\widehat{f}(x) = (x, f(x))$$

Actually, we don't necessarily need all of x, just enough to be able to recover the missing pieces.

Several models of computation can be used to compute reversibly.

Turing Machines
: The "next configuration" has a unique predecessor.

Boolean Circuits
: Circuits that can be turned "upside down."

Cellular Automata
: Discrete dynamical systems that generalize Turing machines.

In all three cases, it takes some effort to make sure that a configuration cannot have multiple predecessors.

Theorem (Bennett 1973)

Any Turing machine can be simulated by a reversible Turing machine. Hence, for any computable partial function f, the inflated function $\widehat{f}$ is reversibly computable.

Needless to say, a standard Turing machine is hopelessly irreversible. To organize a reversible simulation, we proceed as follows:

- First, we introduce a modified type of Turing machine that is naturally locally reversible.
- Then we build a globally reversible simulator using one of these machines that has a history log.

The standard definition of a Turing machine transition function is naturally asymmetric, something like $\delta : Q \times \Gamma \to Q \times \Gamma \times \Delta$ where Q is the state set, Γ the tape alphabet and $\Delta = \{-1, 0, 1\}$ the allowed displacements.

With a view towards reversibility, consider a Turing machine variant where

- The state set Q is partitioned into two parts Q_{write} and Q_{move} of write states and move states.
- The transitions are of the form

$$\delta \subseteq Q \times \Gamma_* \to \Gamma_\Delta \times Q$$

where Γ_* is Γ augmented by a special "don't read" symbol $*$ and $\Gamma_\Delta = \Gamma \cup \Delta$.

Splitting Transitions

Corresponding to the two kinds of states, there are write and move transitions, written

$$pa \to bq \qquad\qquad p* \to dq$$

We can express a standard transition by two split transitions as follows:

$$pa \to qbd \qquad\qquad pa \to bp' \quad p'* \to dq$$

where p' is a new move state (that is not used anywhere else).

The symmetric nature of our split transitions makes it easy to find mutually inverse transitions in the sense that they will revert to the previous configuration from the next one:

$$\begin{array}{ll} pa \to bq & qb \to ap \\ p* \to dq & q* \to -dp \end{array}$$

$$pa \to qb1 \qquad s_1\ s_2\ p\ a\ s_4\ s_5 \quad \rightsquigarrow \quad s_1\ s_2\ b\ q\ s_4\ s_5$$

$$pc \to bp' \qquad s_1\ s_2\ p\ a\ s_4\ s_5 \quad \rightsquigarrow \quad s_1\ s_2\ p'\ b\ s_4\ s_5$$

$$p'* \to 1q \qquad s_1\ s_2\ p'\ b\ s_4\ s_5 \quad \rightsquigarrow \quad s_1\ s_2\ b\ q\ s_4\ s_5$$

This the one-tape situation, but it is easy to generalize to k-tape machines; just consider transitions $p\,\boldsymbol{\sigma} \to \boldsymbol{\tau}\,q$ where $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$ are suitable k-vectors.

For example,

$$p\,(a, b, c) \to (a, d, c)\,q$$

would change the b on the second tape to d, without changing the other tapes. Similarly,

$$p\,(*, *, *) \to (0, 1, 0)\,q$$

would move the tape head on the second tape to the right.

Time to tackle the real problem: we need to enforce our reversibility condition on the configuration graph from above. Bennett's construction uses a 3-tape machine.

Given an ordinary Turing machine $\mathcal{M}$ that computes some function f. We can construct an equivalent, reversible, 3-tape, partitioned machine $\mathcal{M}'$ that works in three phases:

- Run the computation of $\mathcal{M}$ on input x on the work tape and keep a log on the history tape.
- Upon completion, copy the result $f(x)$ from the work tape to the output tape.
- Run the computation backwards, unraveling the history and work tapes, until only x is left on the work tape and the history tape is empty.

Then $\mathcal{M}'$ duly computes $\widehat{f}(x) = (x, f(x))$.

Needless to say, the reversible version of a Turing machine is a bit slower and requires more memory. Surprisingly, the increased resource requirements are quite modest.

Suppose the original machine has time complexity t and space complexity s. For any $\varepsilon > 0$, one can reversibly simulate the machine in

$$t' = O(t^{1+\varepsilon}) \qquad s' = O(s \log t)$$

A different construction minimizing time produces

$$t' = O(t) \qquad s' = O(st^{\varepsilon})$$

This is a bit surprising, reversibility comes relatively cheap. But note that there are hidden constants that increase when ε gets smaller.

Circuits arguably provide a more intuitive framework for reversible computation.

For example, consider standard addition of two bits:

in		out	
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

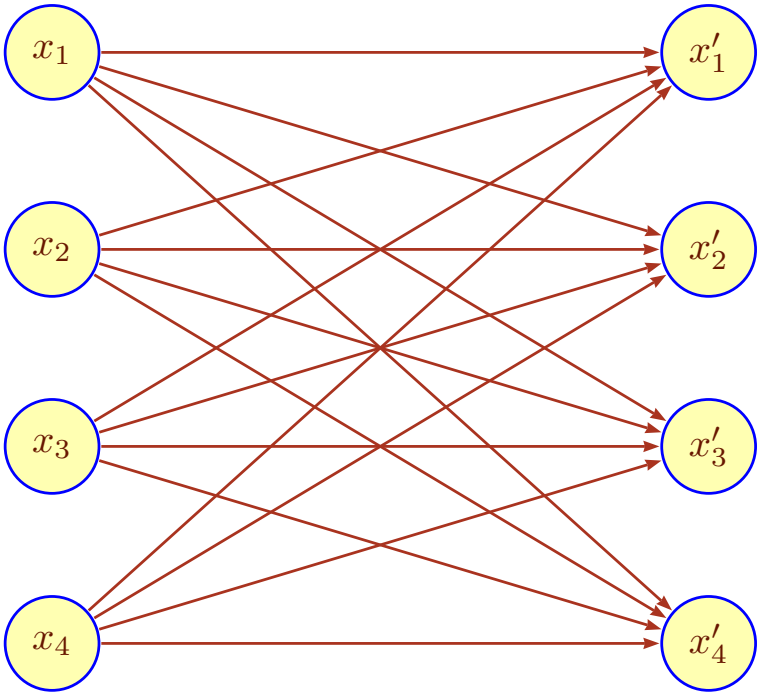
This operation is clearly not reversible: 01 is the result of both inputs 01 and 10, so it is logically impossible to recover the input from the output (regardless of compute-power available).

But we can switch to the inflated version, then everything works out fine.

in		cp		out	
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	1	1	1	0

This operation is trivially reversible though it admittedly looks somewhat clumsy.

Note that we could keep just the green input bit: the other can be determined from the sum. Still, this is not too impressive: we are going from 2 to 3 bits to ensure reversibility.



At least in principle, such a circuit might be reversible.

Needless to say, most standard logic gates couldn't be used in such a circuit. E.g., usual "and" and "or" gates are not reversible for logical reasons, so there is no going back.

Proposition

There are $(2^{2^n})^n$ *Boolean functions* $\mathbf{2}^n \longrightarrow \mathbf{2}^n$.
Only $(2^n)!$ *of these functions are reversible.*

Exercise

Prove the last proposition. Come up with some interesting examples.

For $n = 1$ there is only one interesting reversible n-bit circuit: negation.

For $n = 2$ there are 24 reversible n-bit circuits. Unfortunately, none of them are really interesting. The problem is that they are all affine maps (arithmetic mod 2)

$$\boldsymbol{y} = A \cdot \boldsymbol{x} + \boldsymbol{b}$$

where $\boldsymbol{b}$ is any 2-bit vector, and A is one of the following six matrices:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Reversing the Gates

All these matrices are non-singular over $\mathbb{Z}_2$, so we can compute

$$\boldsymbol{x} = A^{-1} \cdot (\boldsymbol{y} - \boldsymbol{b})$$

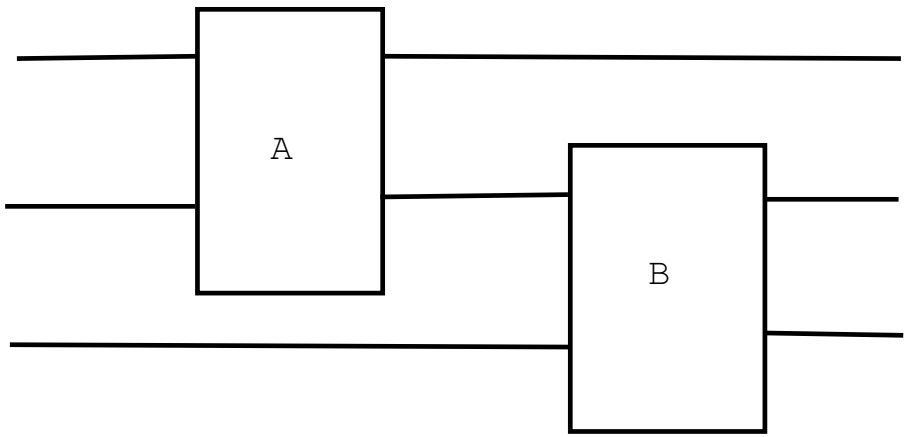
Hence we can recover $\boldsymbol{x}$ from $\boldsymbol{y}$.

For example, with $A = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$ and $\boldsymbol{b} = \mathbf{0}$ we get

x	y	x'	y'
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

This is the CNOT gate: x is a control bit and y is flipped if $x = 1$.

We might try to get more interesting maps $\mathbf{2}^k \to \mathbf{2}^k$ by composing several of these two-input 2 circuits.



However, affine maps are closed under composition, so using these gates as components we cannot implement anything other than affine maps in any dimension.

But $n = 3$ becomes really interesting. Somehow there are enough degrees of freedom to do non-trivial reversible computation now.

For example, there is the Toffoli gate or CCNOT gate:

x	y	z	x'	y'	z'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Here x and y are control bits (so $x' = x$, $y' = y$) and z is flipped provided both x and y are 1: $z' = z \oplus (x \wedge y)$.

The Toffoli gate is universal in the sense that one can construct a NAND and a copy gate from it. From these, we can build a universal computer.

To construct a NAND gate, clamp $z = 1$.

x	y	z	x'	y'	z'
0	0	1	0	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

For copy, clamp $x = 1$ and $z = 0$ so that $z' = y$.

x	y	z	x'	y'	z'
1	0	0	1	0	0
1	1	0	1	1	1

Fredkin Gate

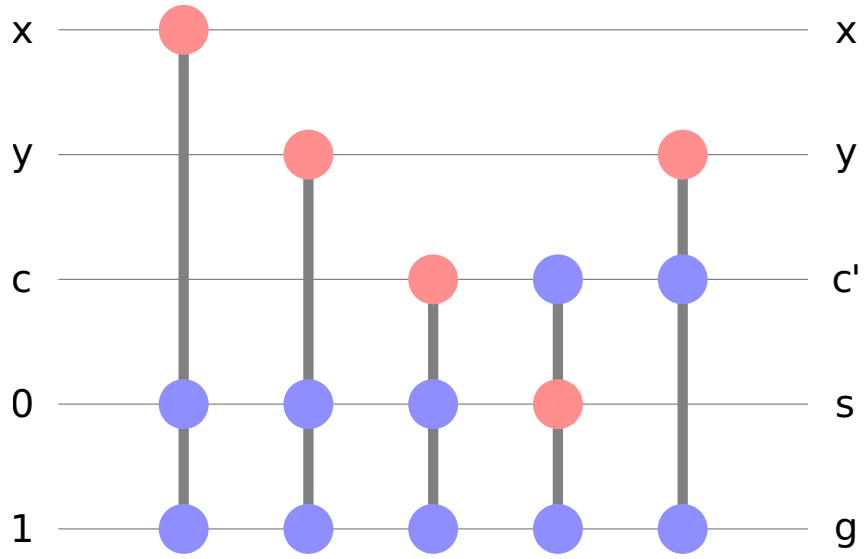
Here is another reversible $n = 3$ gate: x is a control bit, and y and z are swapped provided that $x = 1$.

x	y	z	x'	y'	z'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

Exercise

Figure out how to implement logic using Fredkin gates.

Fredkin Adder



Input bits: p, q, r plus $p = 0$ and $g = 1$.

5 rounds of applying a Fredkin gate

Output: p, q, carry-bit, parity-bit, g (a garbage bit)

Building reversible circuits in silicone is clearly technologically the most attractive way to implement reversible computation. Ideally, one would like to have the whole CPU work reversibly, but it is of interest to handle just parts of the whole device reversibly. For example, reversible arithmetic logic units have been designed.

At any rate, quantum computing has taken over, at least for the time being.

This year, there is the 15th annual Conference on Reversible Computation.

It is my opinion that everything must be based on a simple idea. And it is my opinion that this idea, once we have finally discovered it, will be so compelling, so beautiful, that we will say to one another, yes, how could it have been any different.

John Archibald Wheeler

J. A. Wheeler had a radical proposal: all of physics reduces to a bunch of bits (aka the importance of information theory for physics).

- We can only ask 1-bit question from nature.
- We reconstruct reality from the answers to moderately large numbers of such 1-bit questions.

Incidentally, he also coined the terms black hole, quantum foam and wormhole.

“Calculating Space”
Konrad Zuse (1969)

Ed Fredkin

There is a nice model of computation that is physics-friendly and makes reversibility almost visible to the naked eye: cellular automata.

In the easiest case, a cellular automaton consist of a two-way infinite one-dimensional grid of cells. At each time t, each cell is in a state in $\mathbf{2}$.

This represented by a configuration $X : \mathbb{Z} \to \mathbf{2}$.

Here comes the important part: each configuration X leads to a new configuration $F(X)$ according to a simple, local update rule. We apply the update rule to all overlapping blocks of 3 consecutive bits in X, in parallel.

Rinse and repeat.

Technically, the update rule is given as a local map (local rule), a Boolean function

$$f : \mathbf{2}^3 \to \mathbf{2}$$

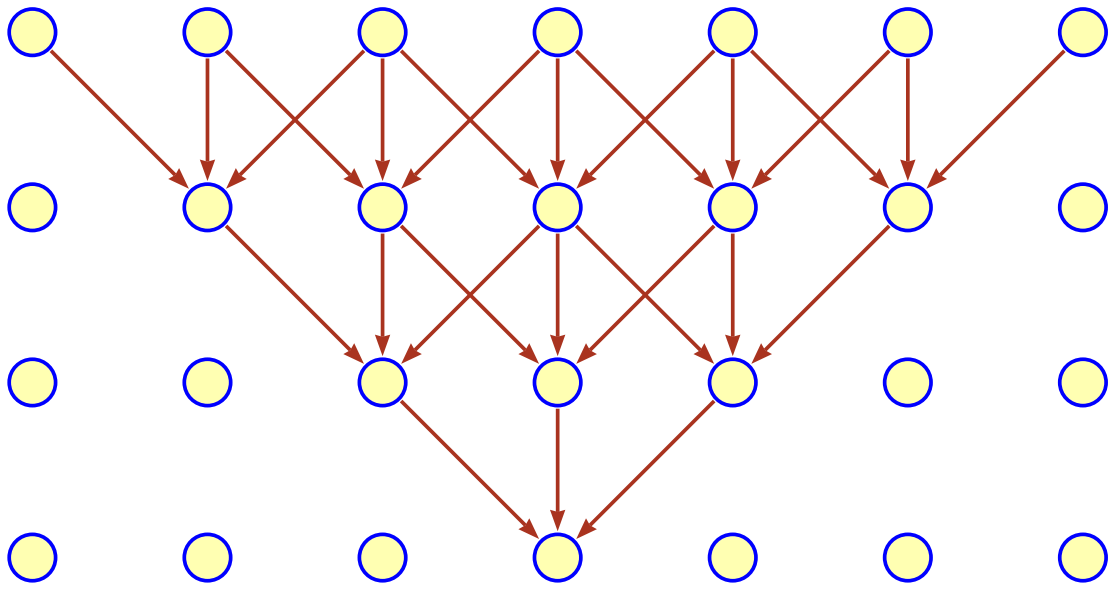
It extends to the global map (global rule) via

$$F(X)(i) = f(X(i-1), X(i), X(i+1))$$

To get the new bit at position i, we collect the current bit at position i, plus its left and right neighbor, and apply the local rule.

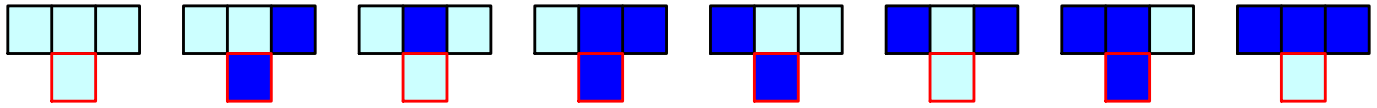
Note that there are only $2^{2^3} = 256$ local rules, so one can easily try them all out. Each rule is specified by a single byte, so we can think of them as being numbered 0 to 255.

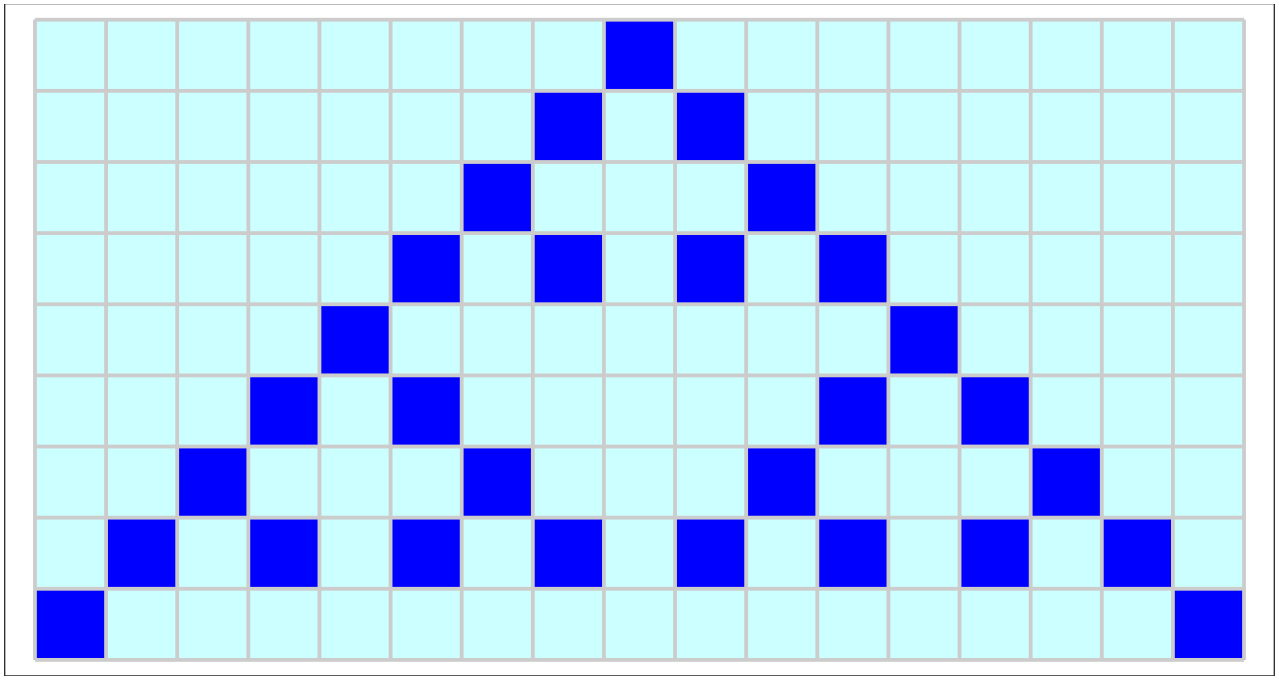
A Homogeneous Circuit

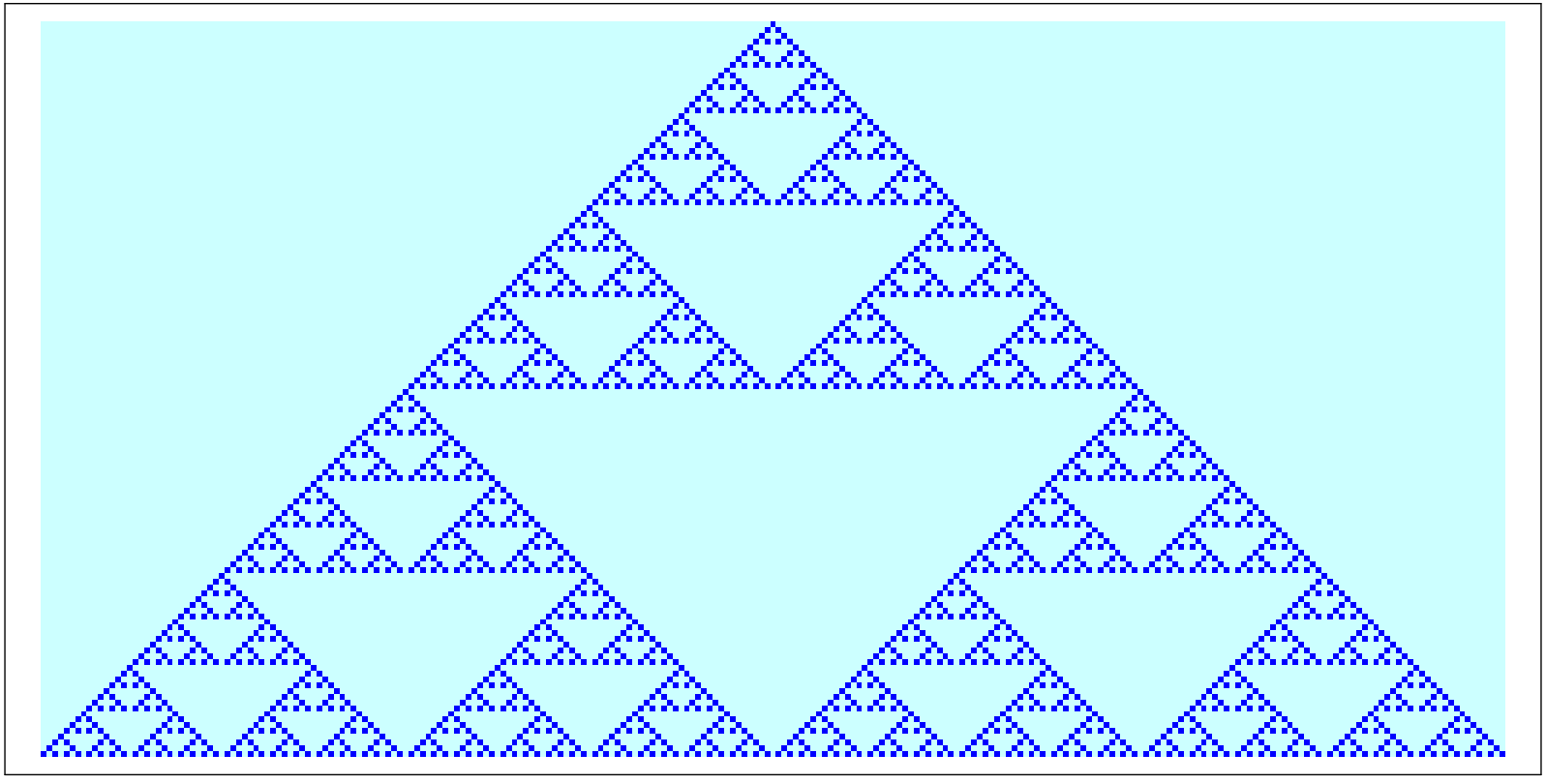


Example: Rule 90

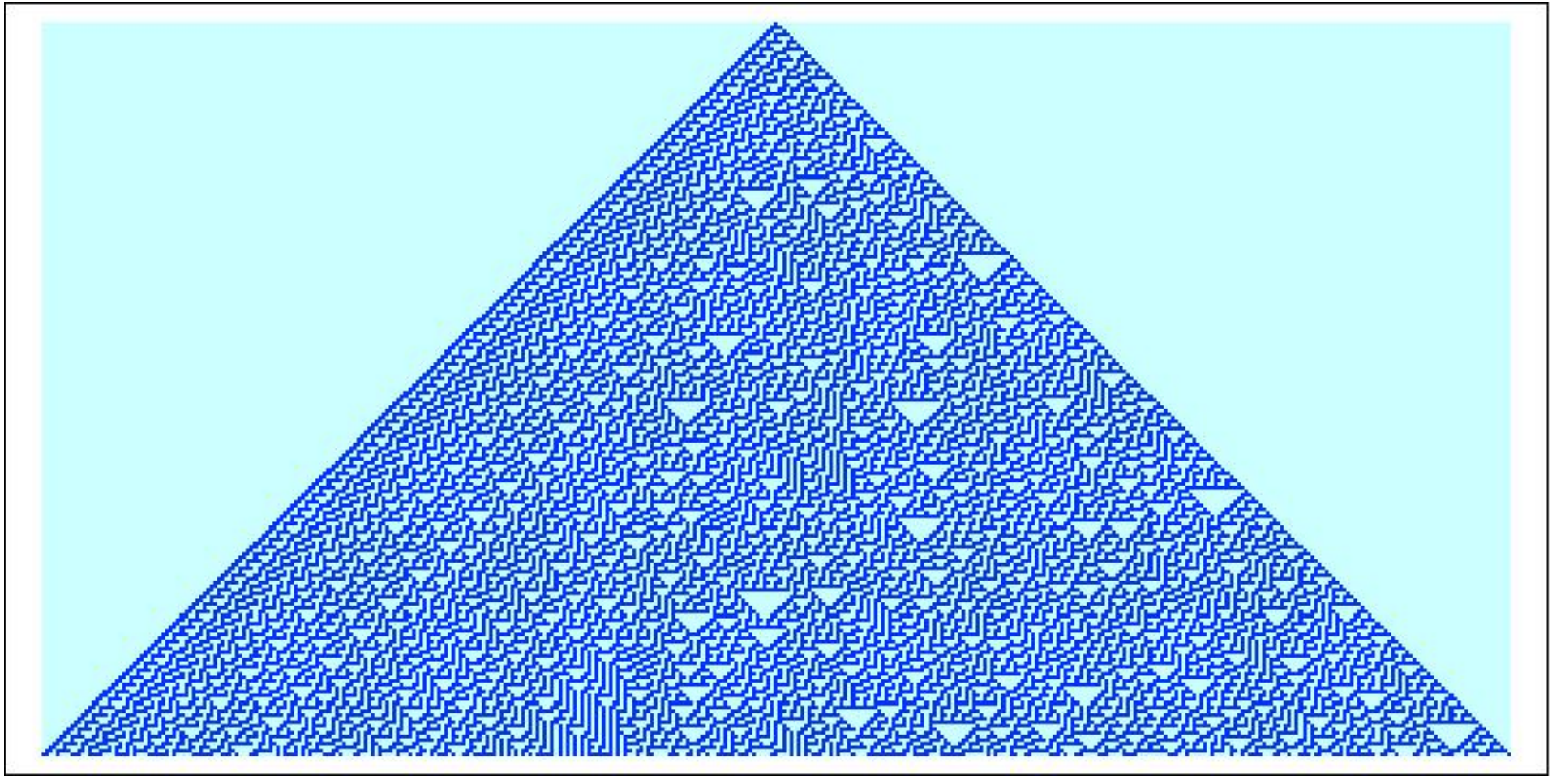
Here $f(x, y, z) = x \oplus z$.

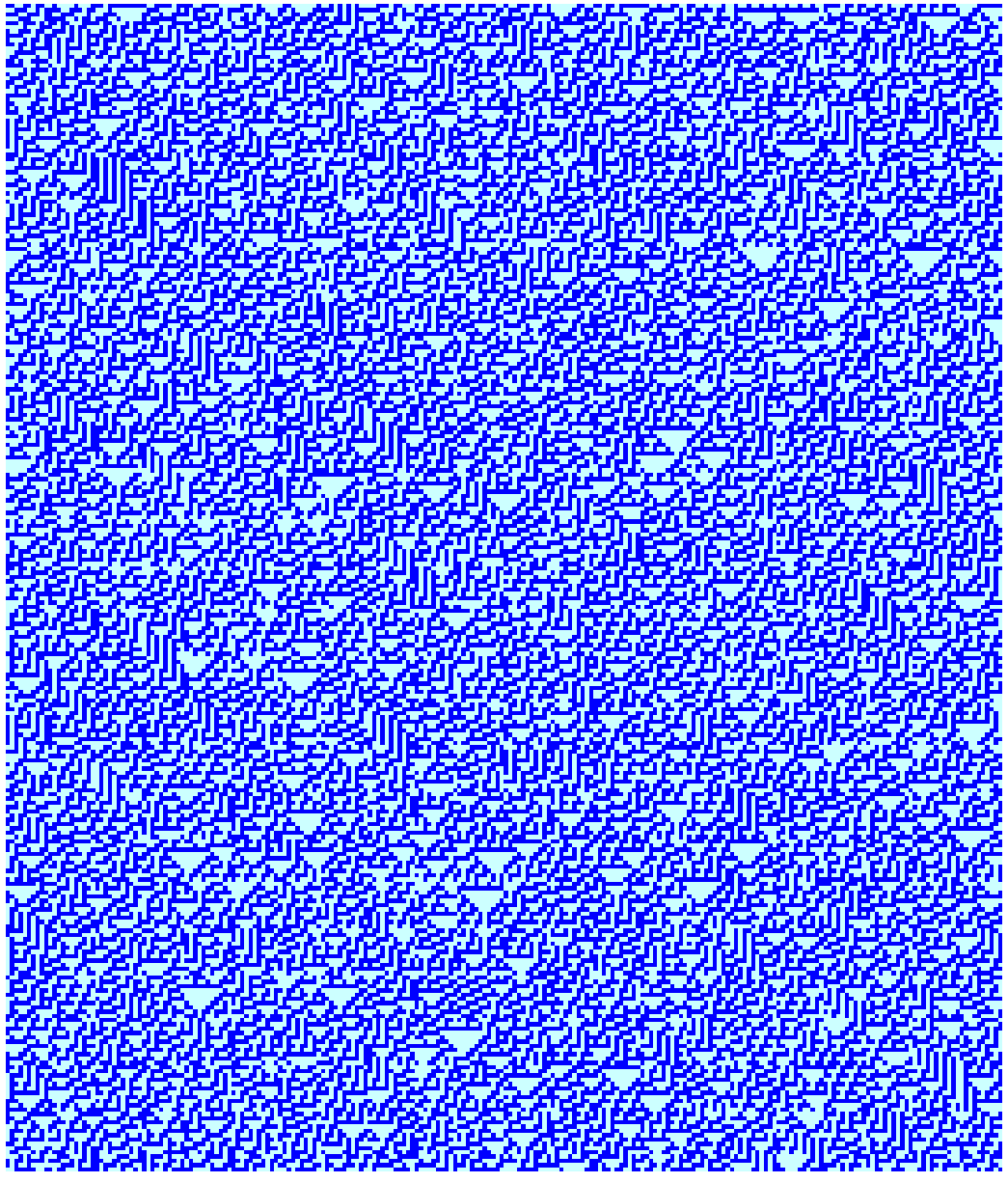




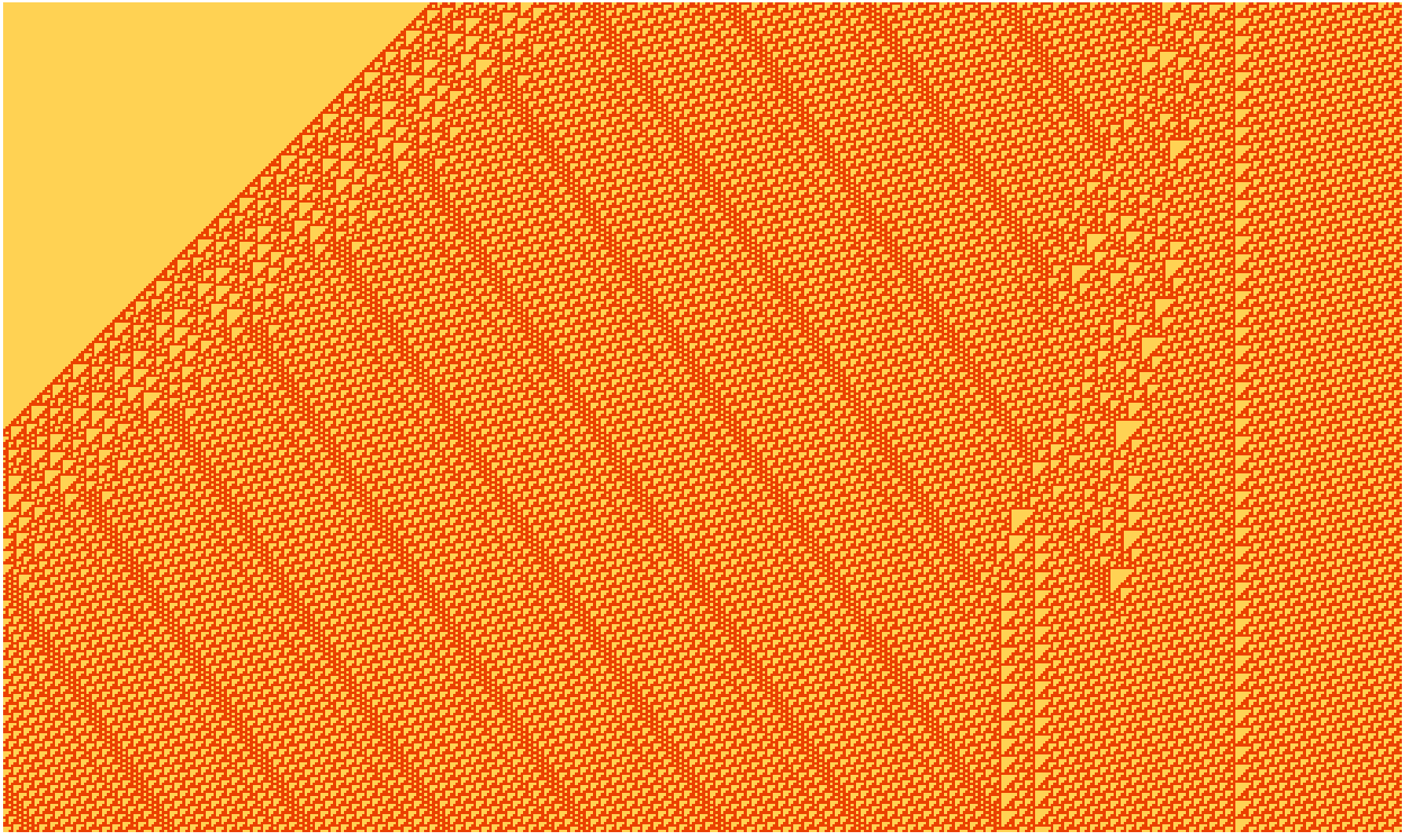


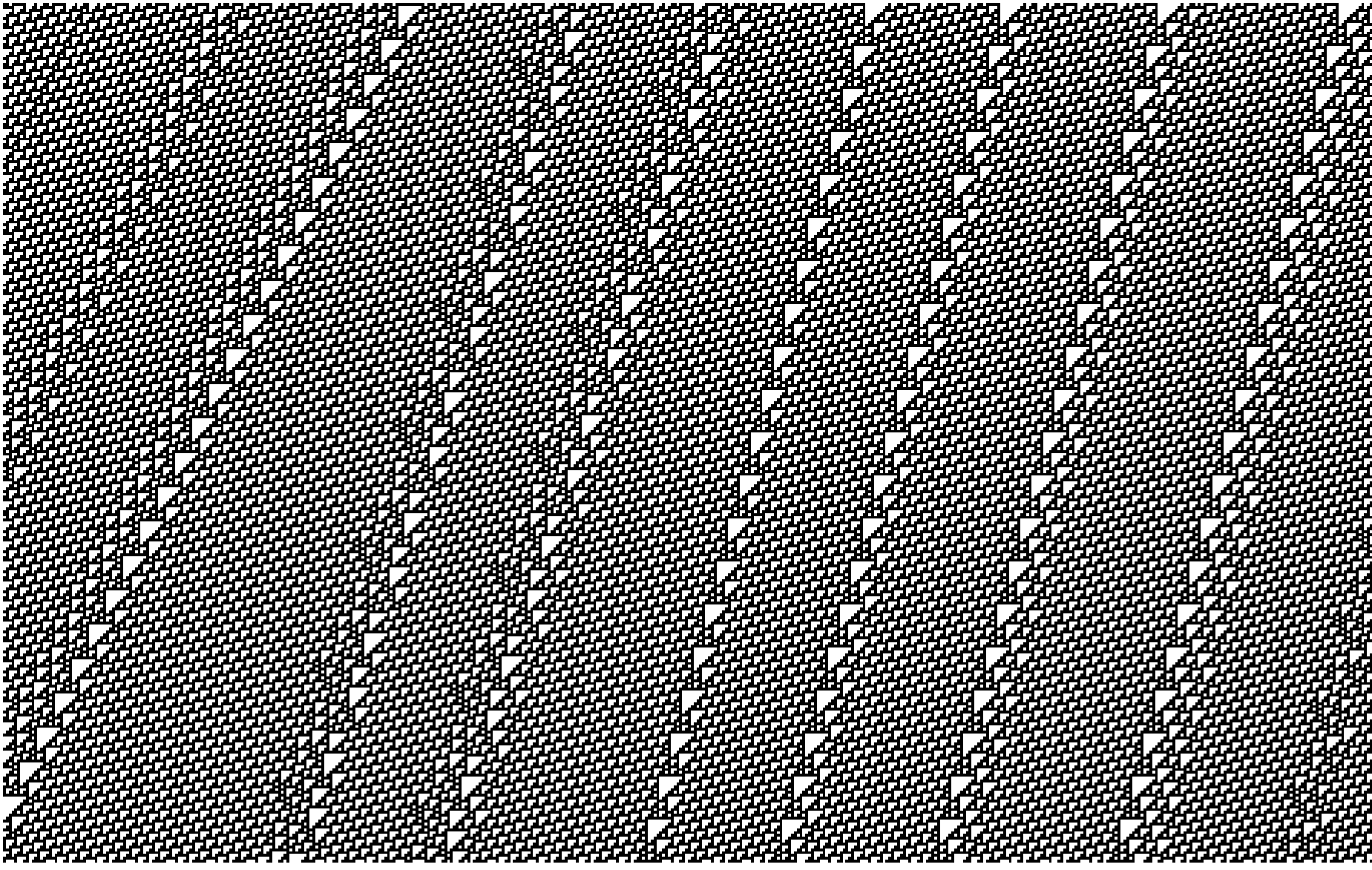
Example: Rule 30





Example: Rule 110



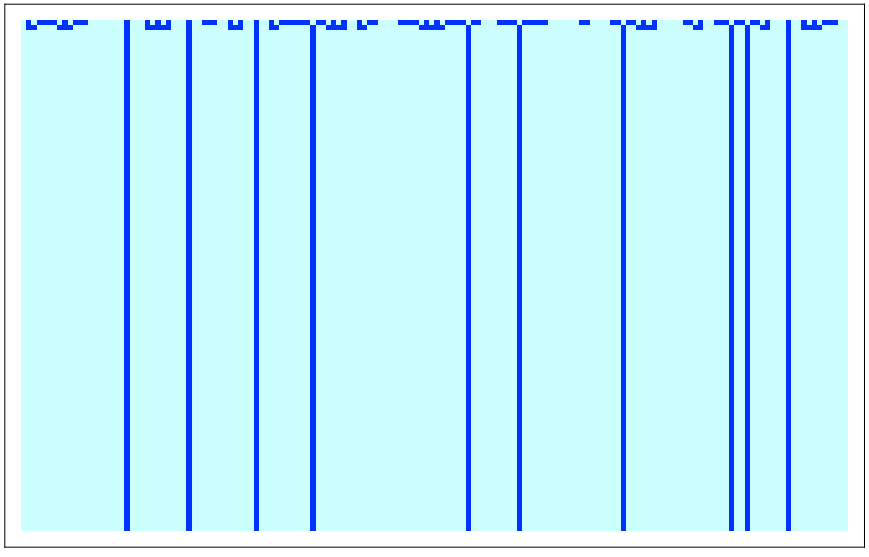


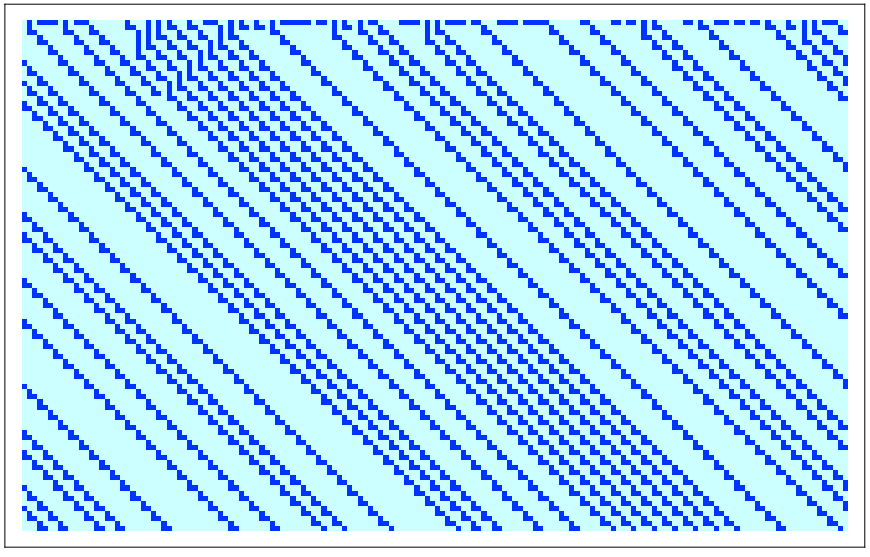
Theorem (M. Cook)

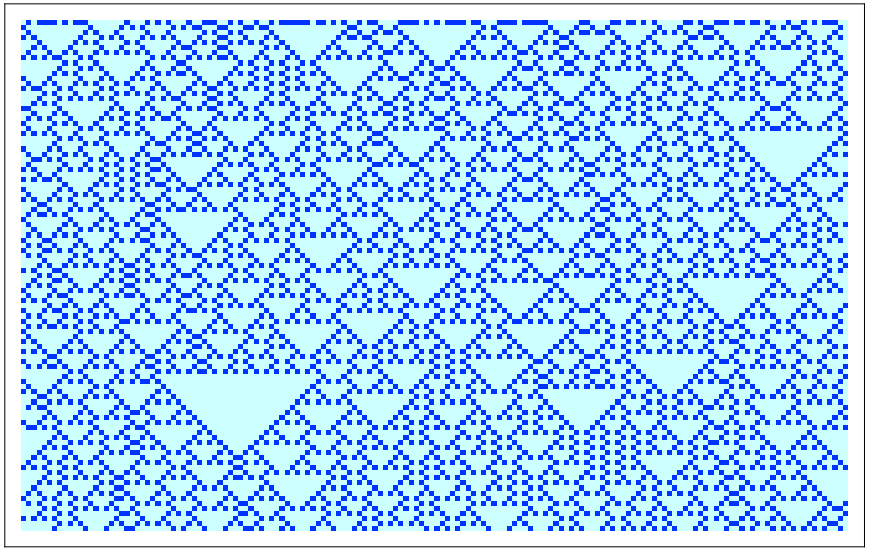
Rule 110 is computationally universal.

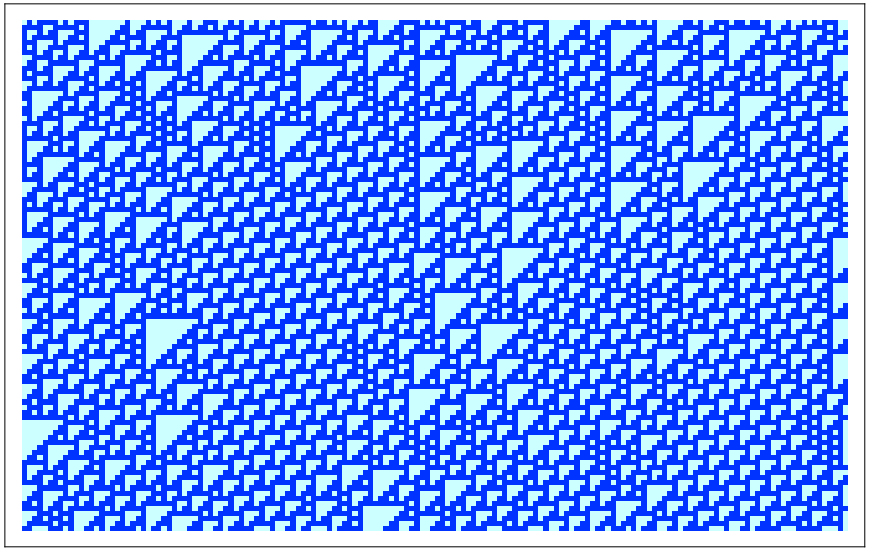
This requires a highly sophisticated simulation of 2-tag systems.

All current "proofs" are heavy on pictures, it would be a great project to get this through a theorem prover.



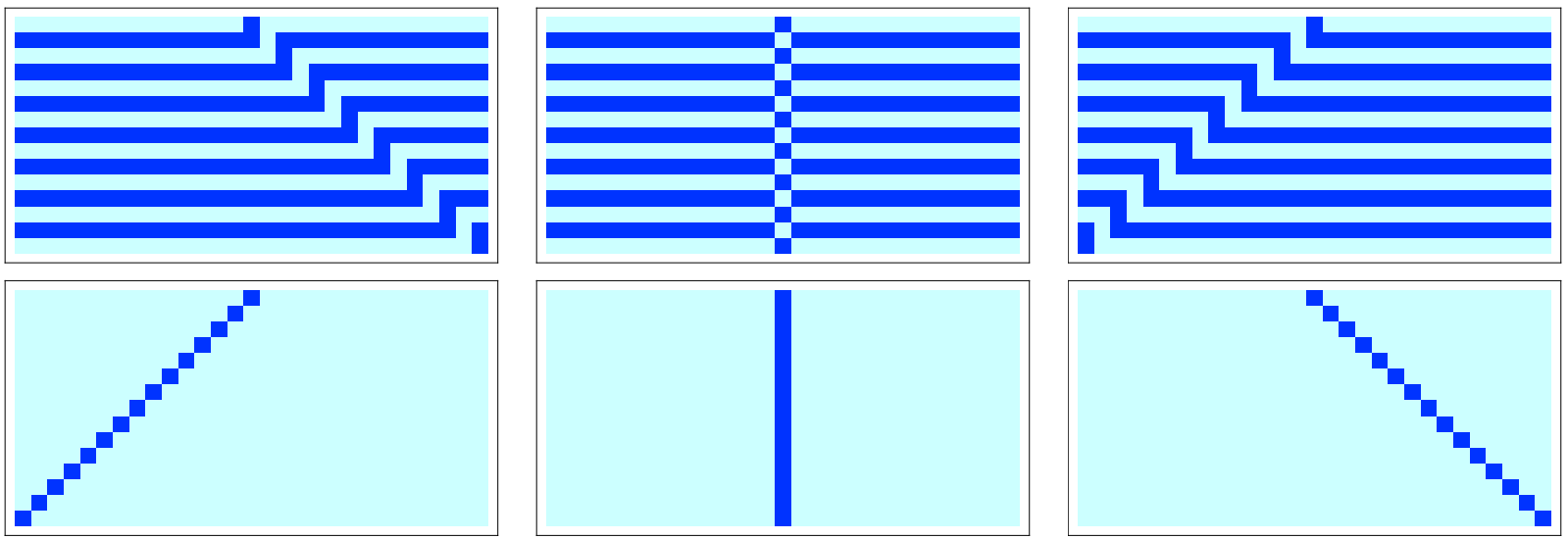






The construction of reversible Turing machines is somewhat involved, but for 1-dimensional cellular automata there are several nice ways to produce reversibility.

Alas, there are no interesting examples of reversible elementary CA.



Bit-flipping, shifts and identity.

There are several ways to generalize our basic model of cellular automata.

Alphabet Each cell carries a symbol from Σ.

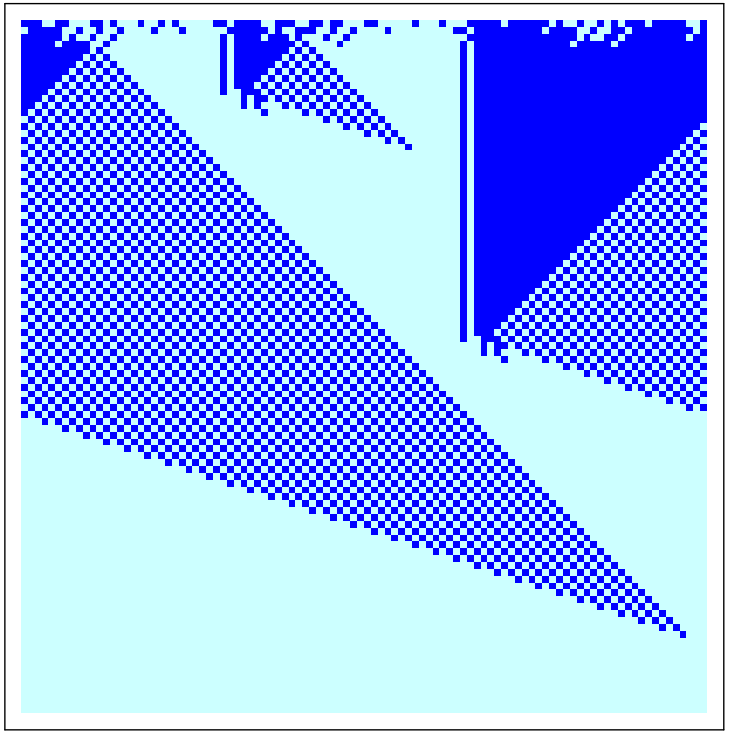
Neighborhood Consider all neighbors up to distance r.

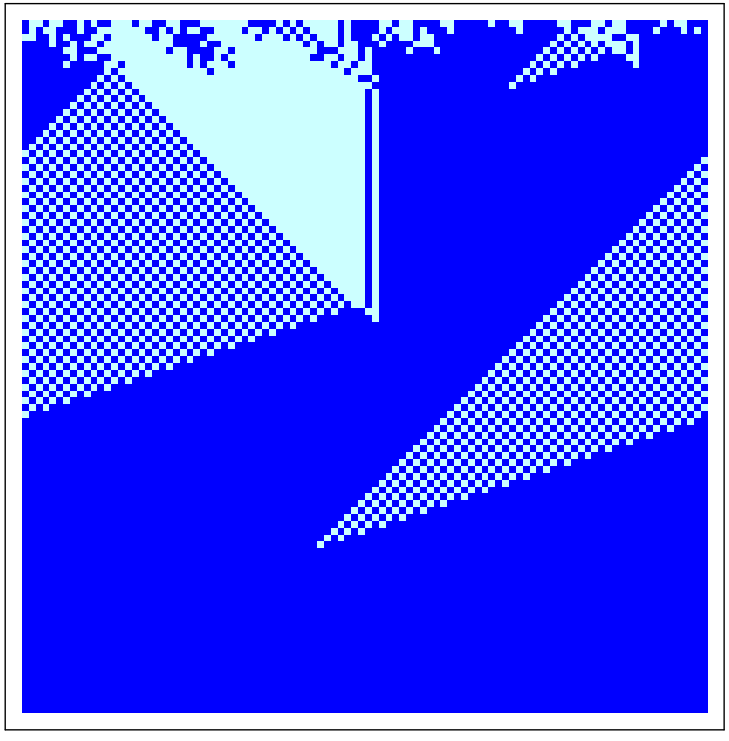
Dimension Work on a d-dimensional grid.

So the local map then looks like

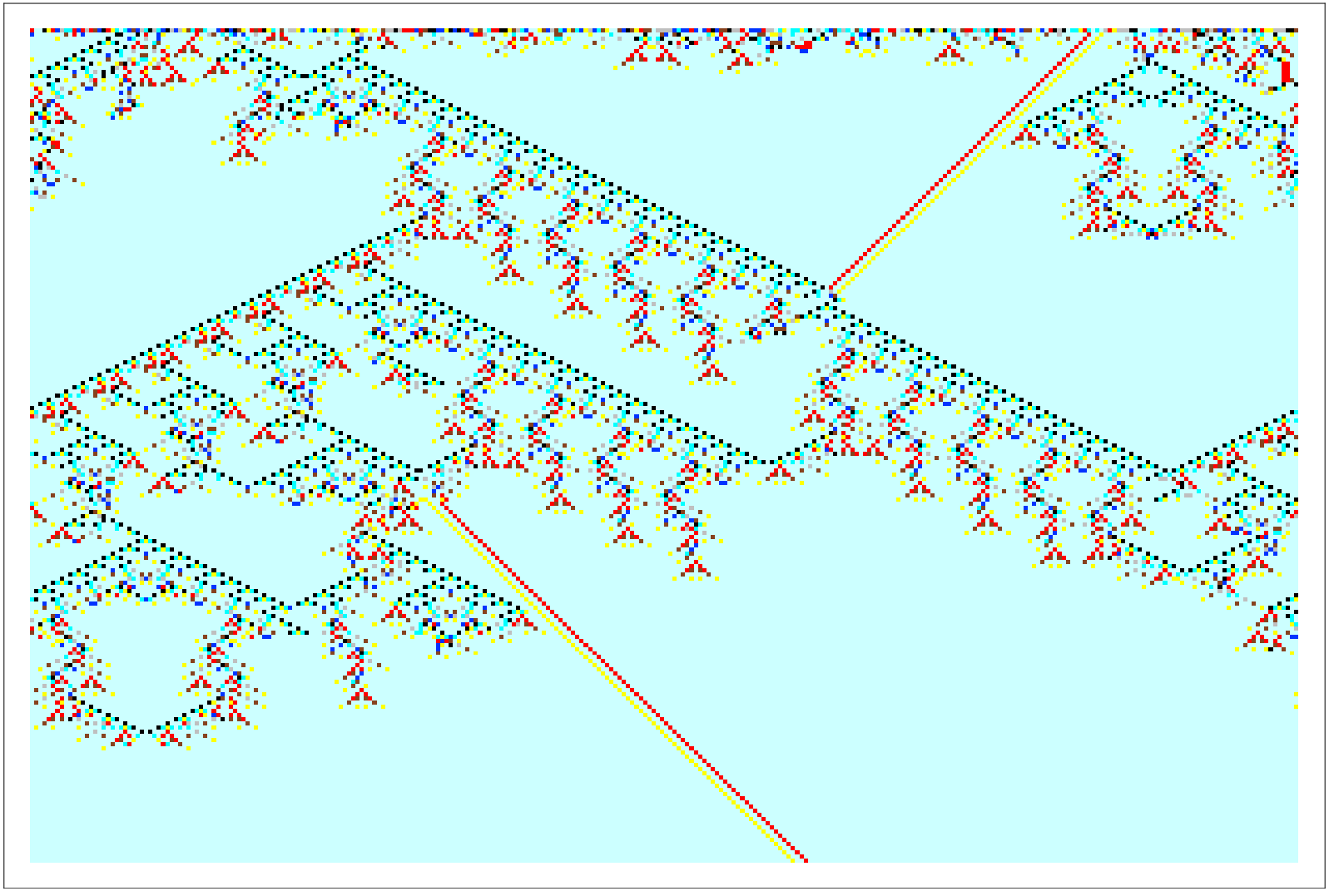
$$\Sigma^{[-r,r]^d} \to \Sigma$$

Example: Majority





Example: Langton's CAs



Here is a wild way of constructing reversible CAs, based on the idea of second order systems: the next state X_{n+1} depends on both X_n and X_{n-1}.

Suppose configurations are binary. Given a binary CA F consider the recurrence

$$X_{n+1} = F(X_n) \oplus X_{n-1}$$

Note that $X_{n-1} = F(X_n) \oplus X_{n+1}$ so we can run the recurrence backwards.

A moment's thought reveals that this recurrence can be handled by a cellular automaton, albeit over a larger alphabet.

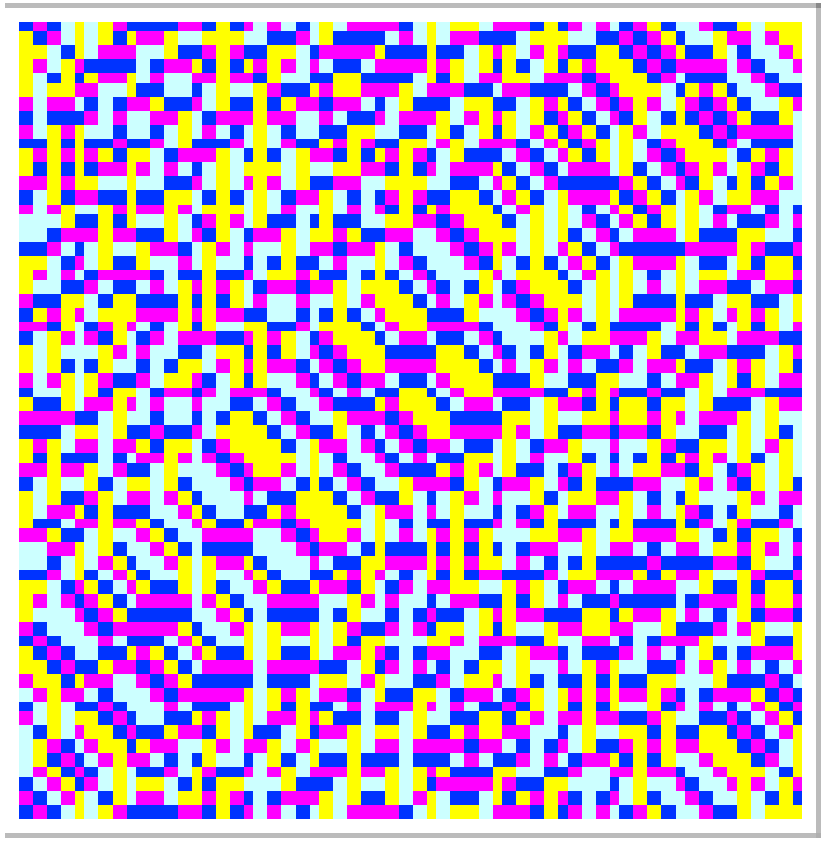
Let $\Sigma = \mathbf{2} \times \mathbf{2}$: each configuration is a pair of binary configurations (think convolution): the upper track holds X_{n-1}, the lower X_n of the old CA.

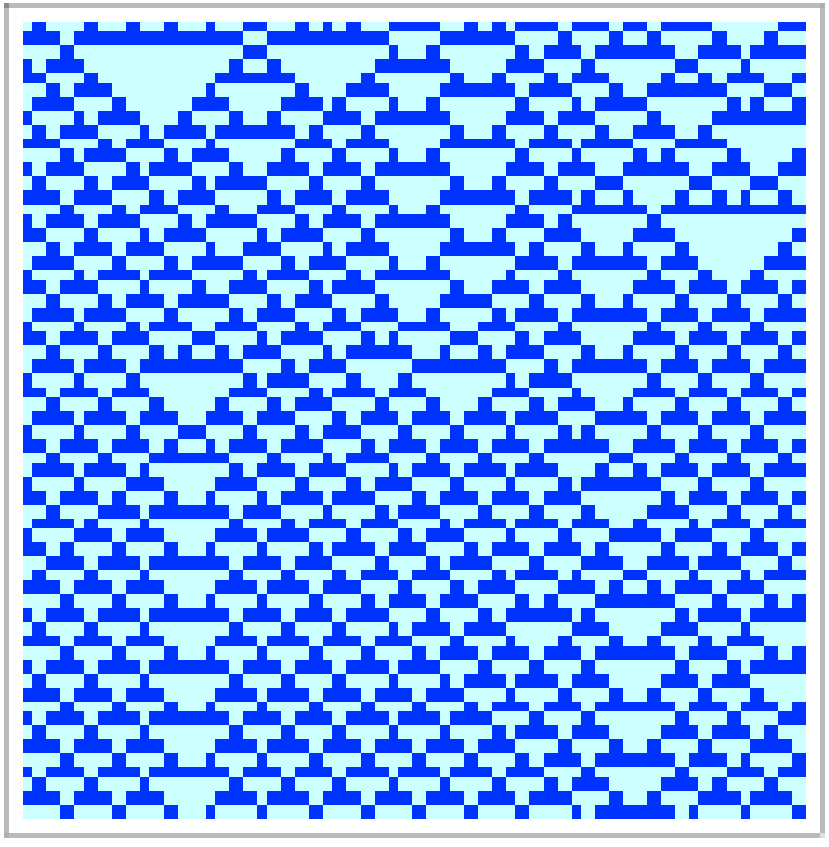
The update rule of the new CA is

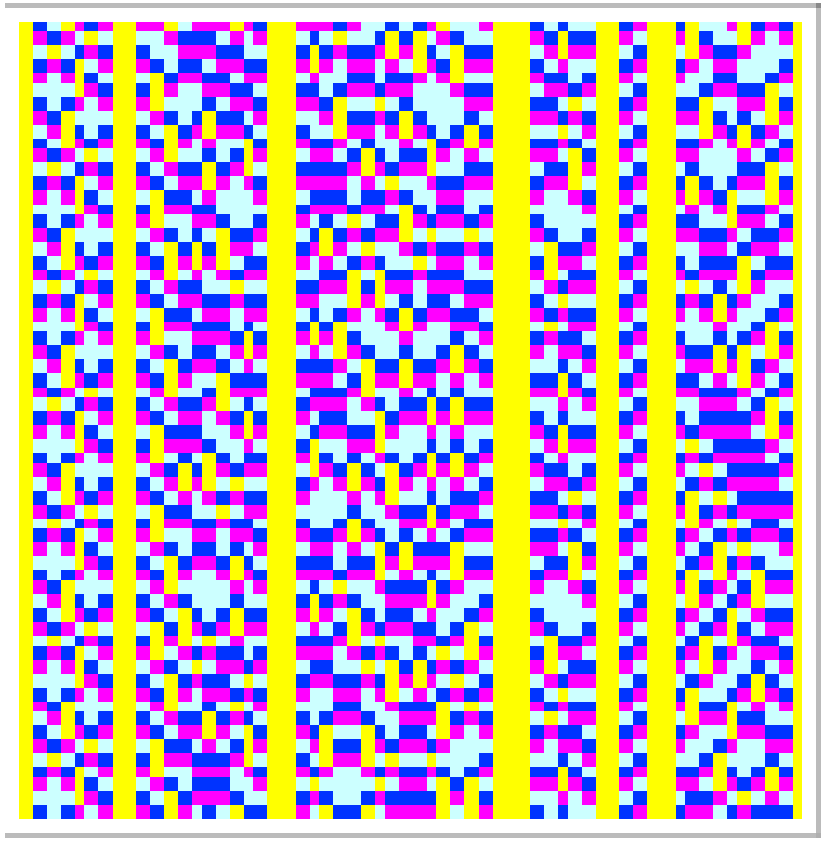
$$G(X, Y) = (Y, X \oplus F(Y))$$

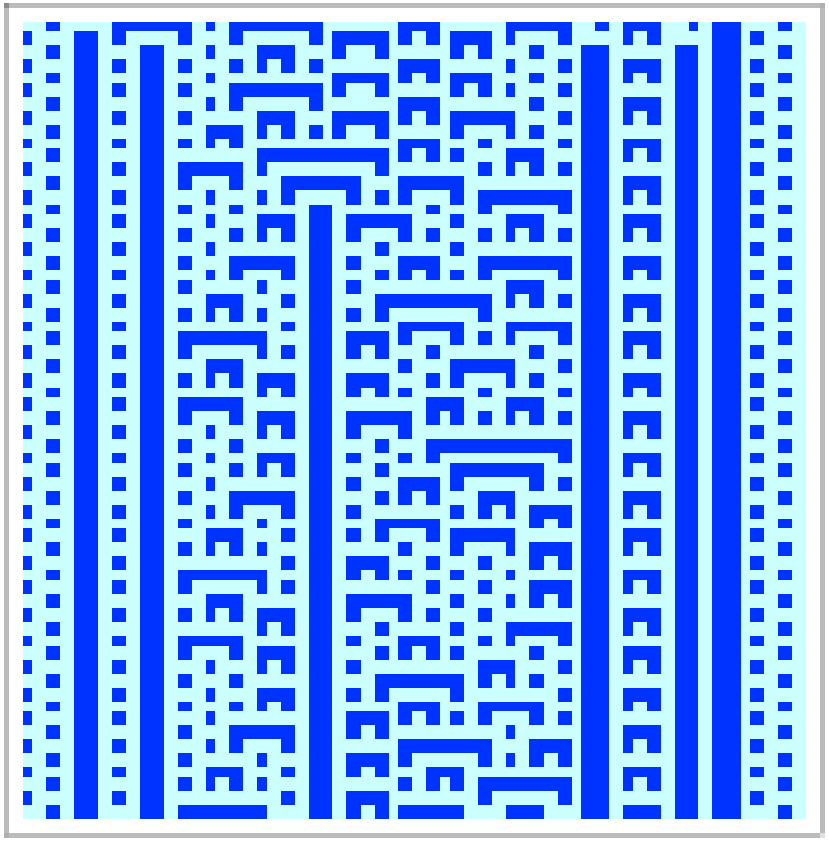
Claim

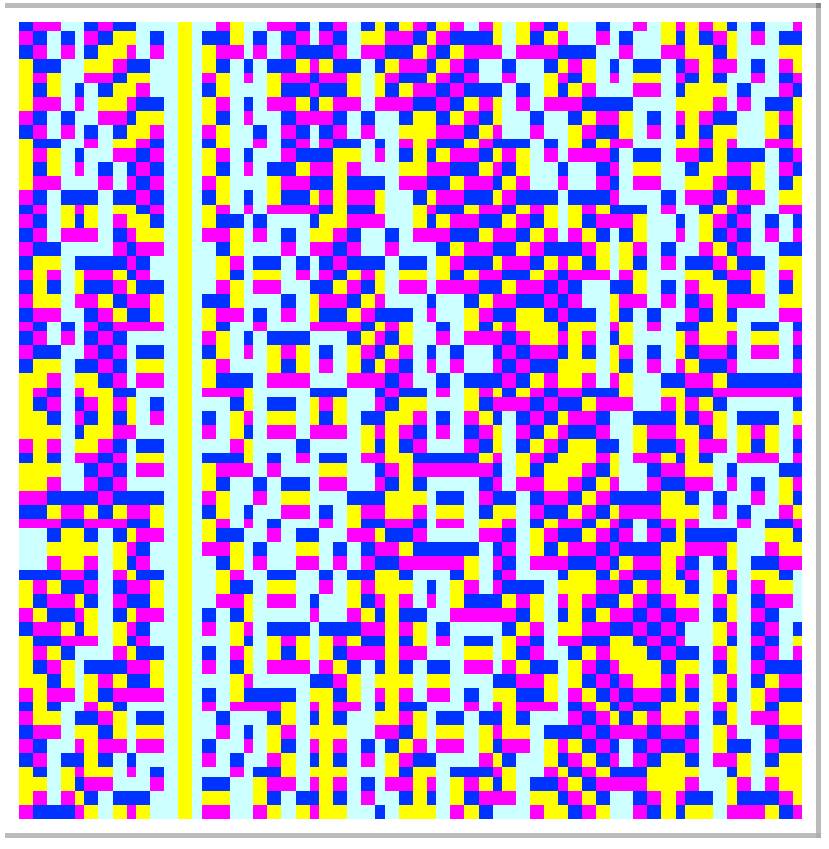
G is reversible (no matter whether F is).

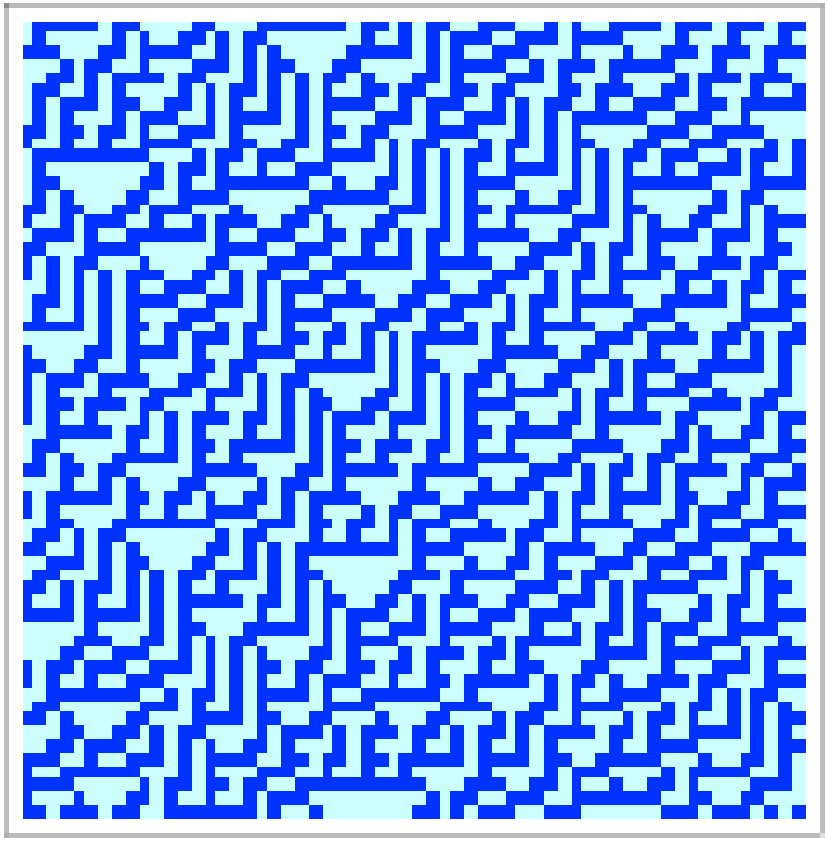


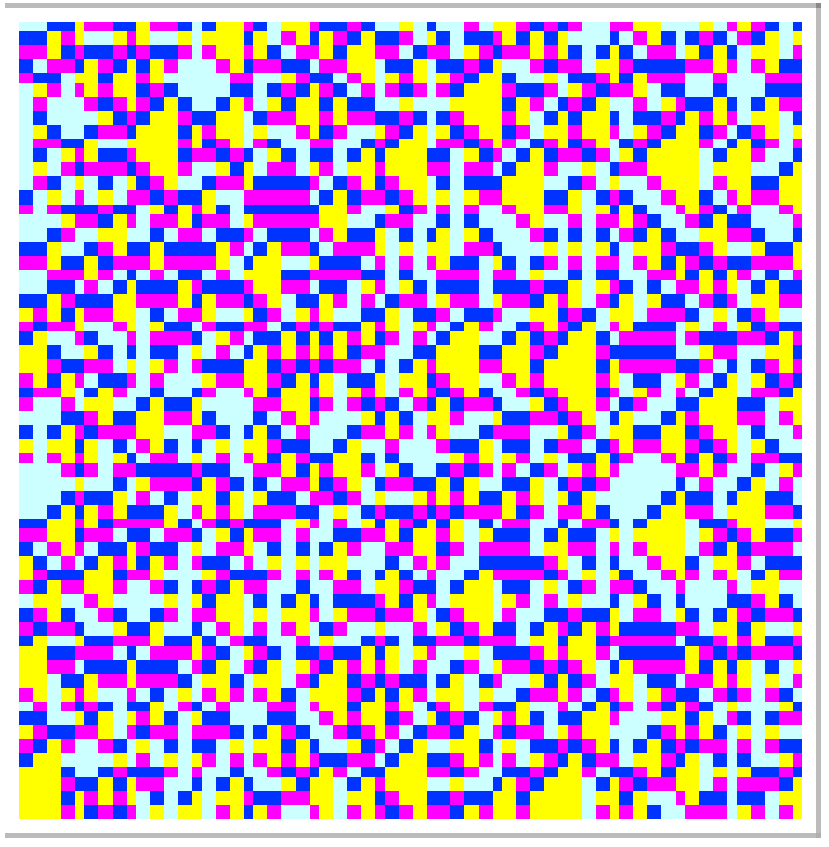


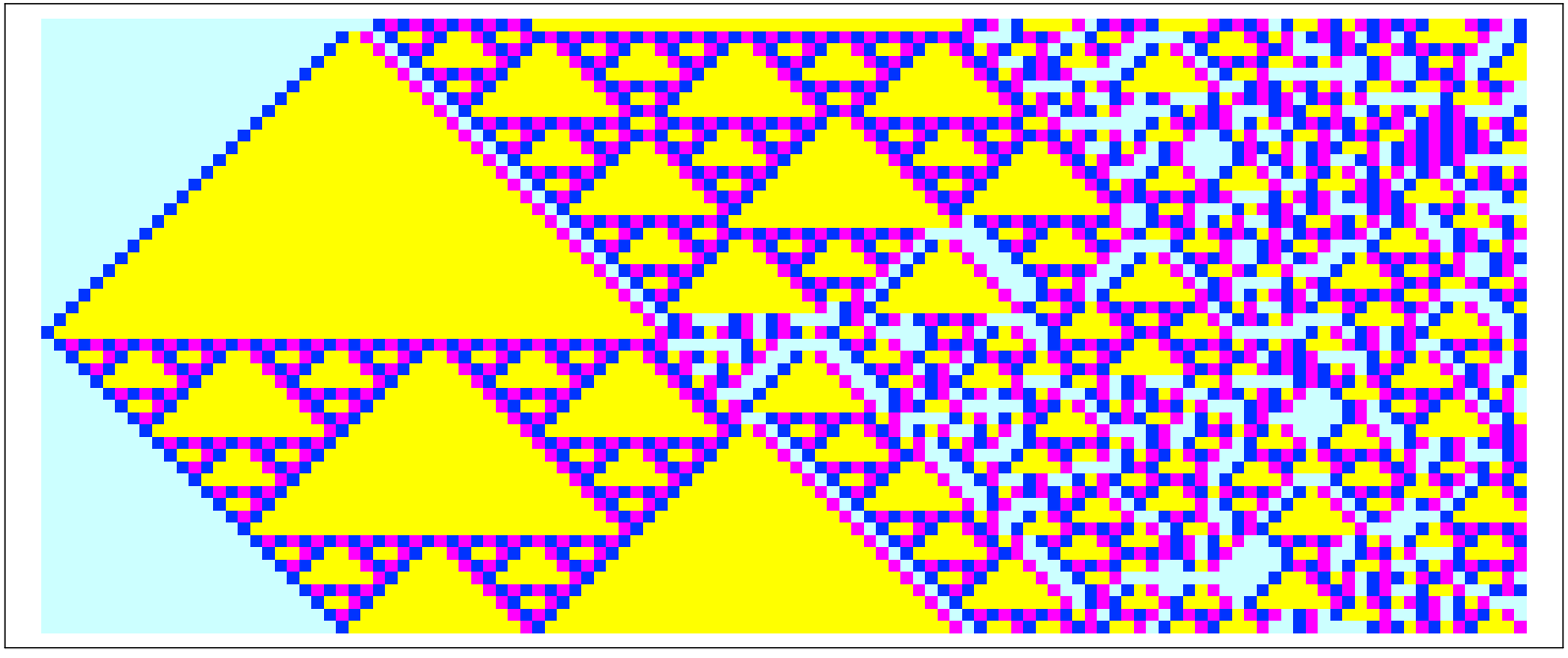


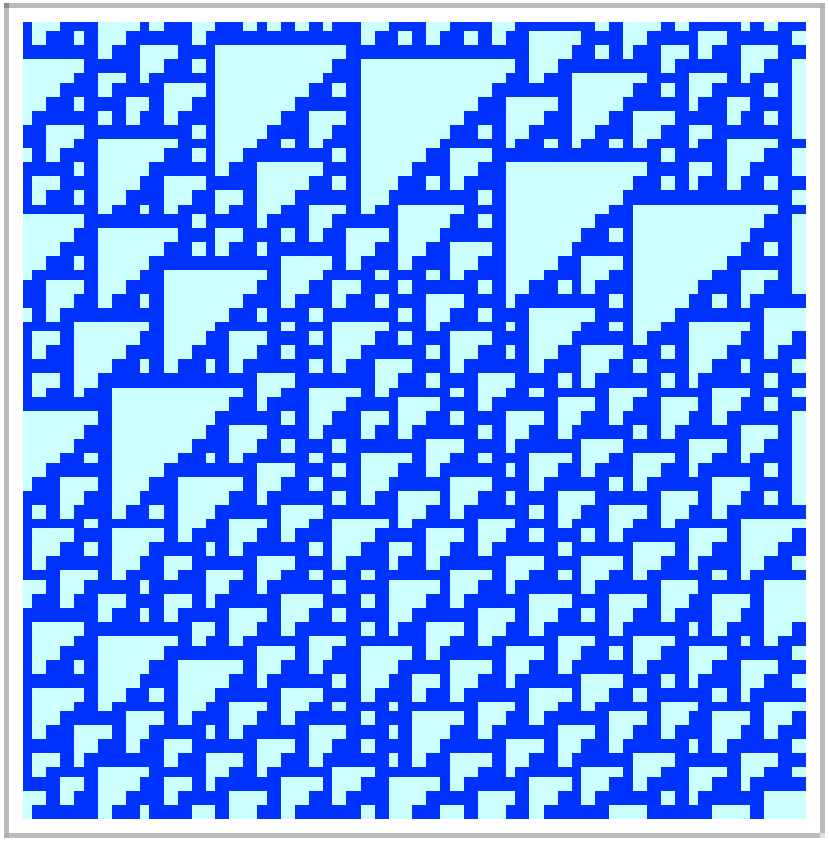


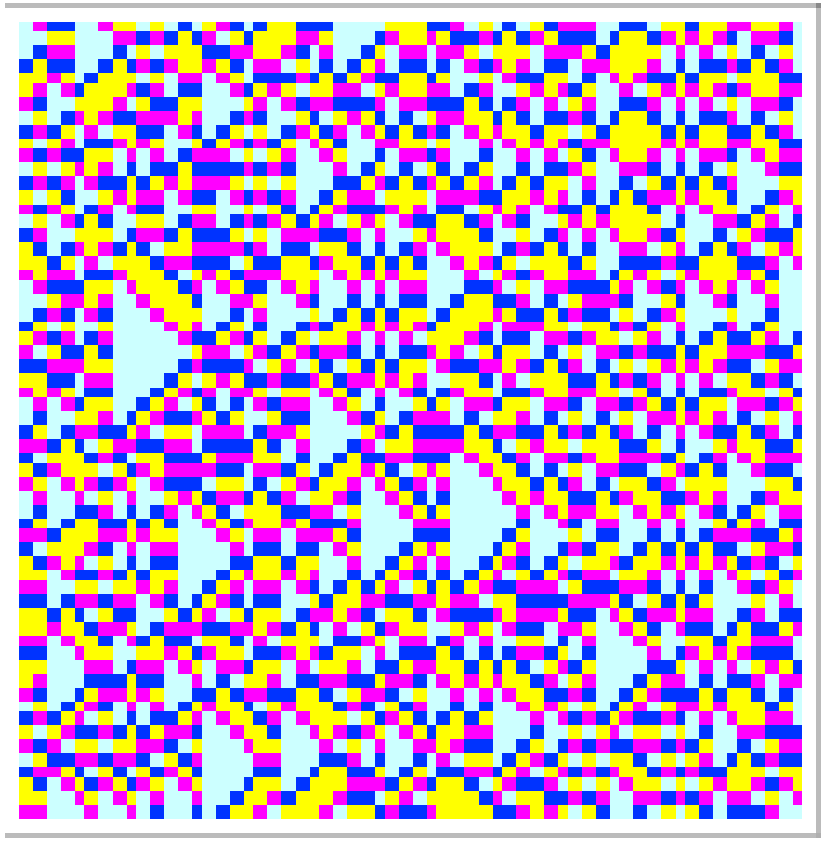


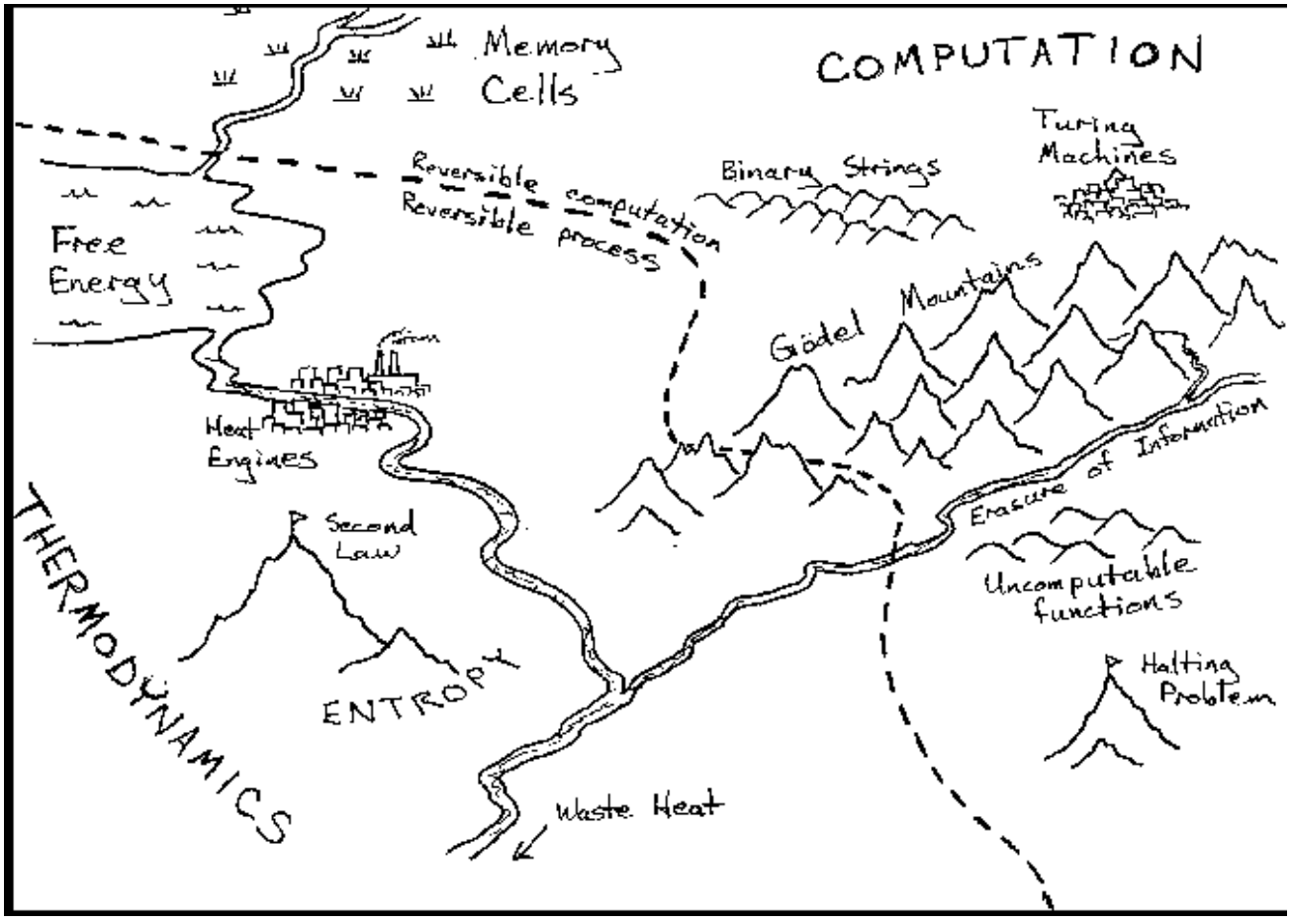












Stolen from somewhere on the web, with apologies. Ponder deeply . . .