P vs. NP,
Phase Transitions,
and Incomputability in the Wild

Cristopher Moore
Santa Fe Institute

An analogy

A set *S* is **decidable** if there is an algorithm* *A*(*x*) that runs in finite time, such that

$$x \in S \Leftrightarrow A(x) = 1$$

A set *S* is **recursively enumerable (RE)** if there is an algorithm *A*(*x*,*w*) that runs in finite time such that

$$x \in S \Leftrightarrow \exists w : A(x, w) = 1$$

e.g. a Turing machine *x* halts if there is an integer *t* such that *x* halts after *t* steps. *A*(*x*,*t*) runs *x* for *t* steps and checks that *x* halts.

*you know what I mean.

An analogy

A set *S* is in **P** if there is an algorithm *A*(*x*) that runs in poly(*n*) = O(n^c) time (where *n* is the length of *x*) such that

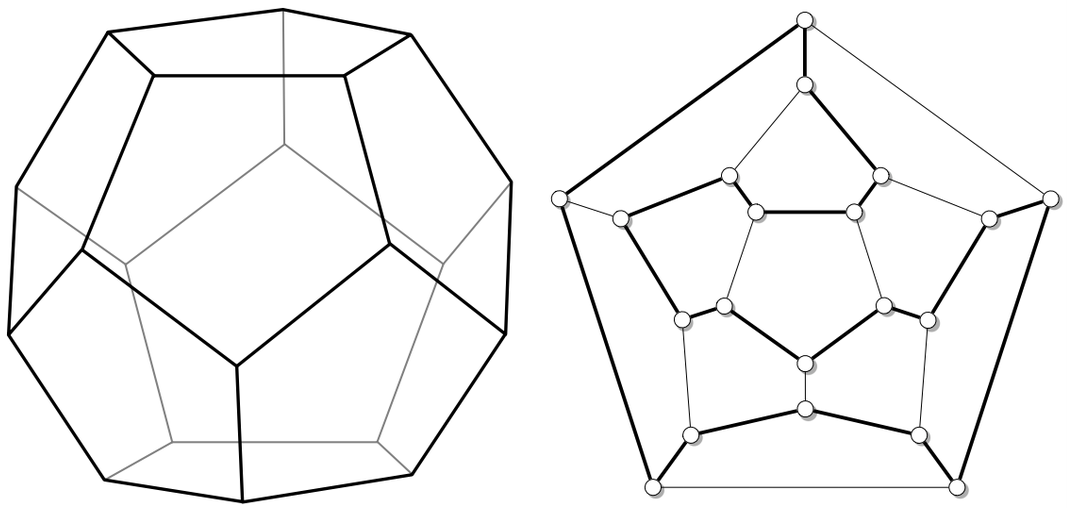
$$x \in S \Leftrightarrow A(x) = 1$$

A set *S* is in **NP** if there is an algorithm *A*(*x*,*w*) that runs in poly(*n*) time, such that

$$x \in S \Leftrightarrow \exists w : A(x, w) = 1$$

e.g. a graph *x* is Hamiltonian if there is a tour *w* that visits each vertex once. *A*(*x*,*w*) checks that *w* is a Hamiltonian tour of *x*.

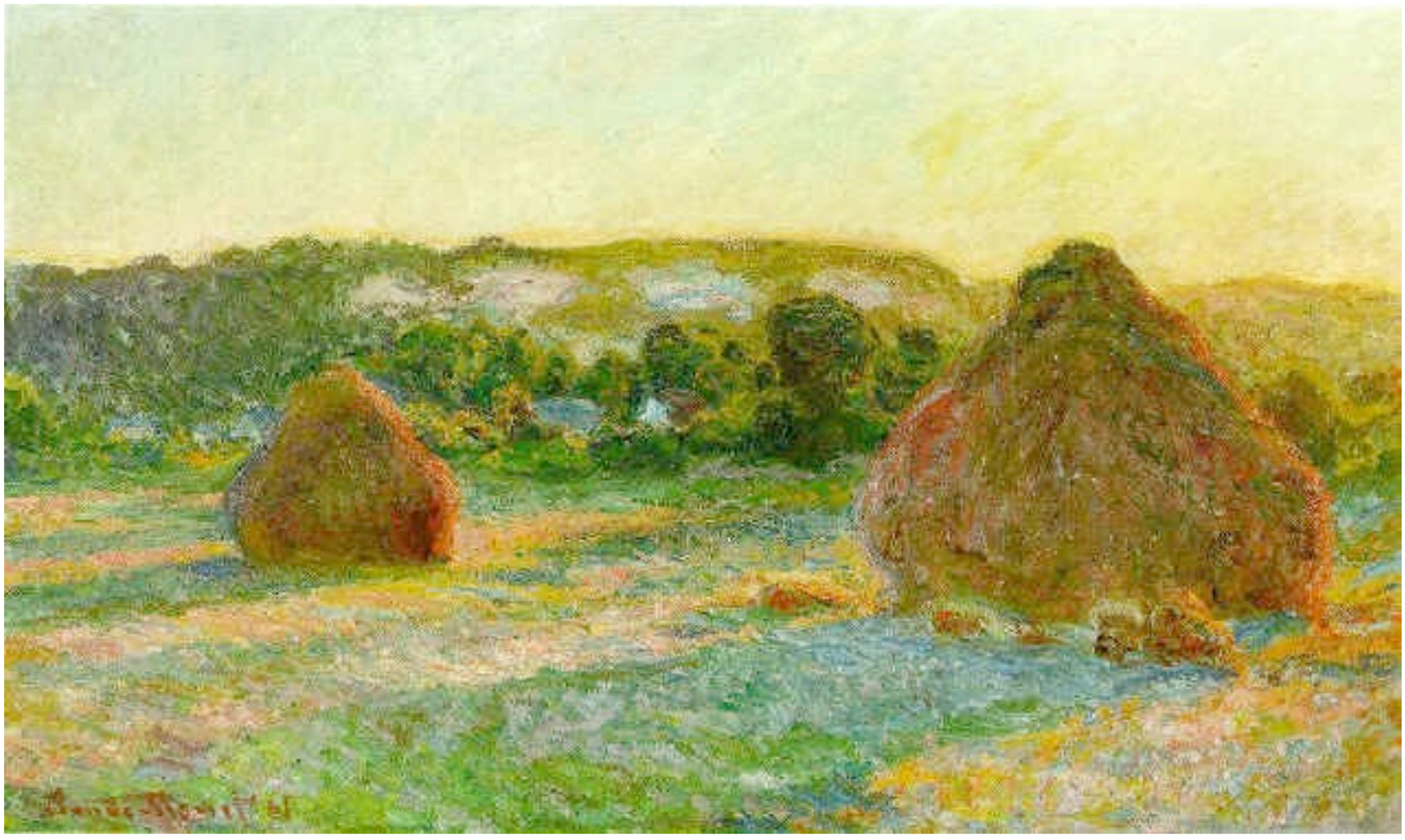
Note asymmetry! There is an easy proof if the answer is “yes”, but not necessarily if it is “no”



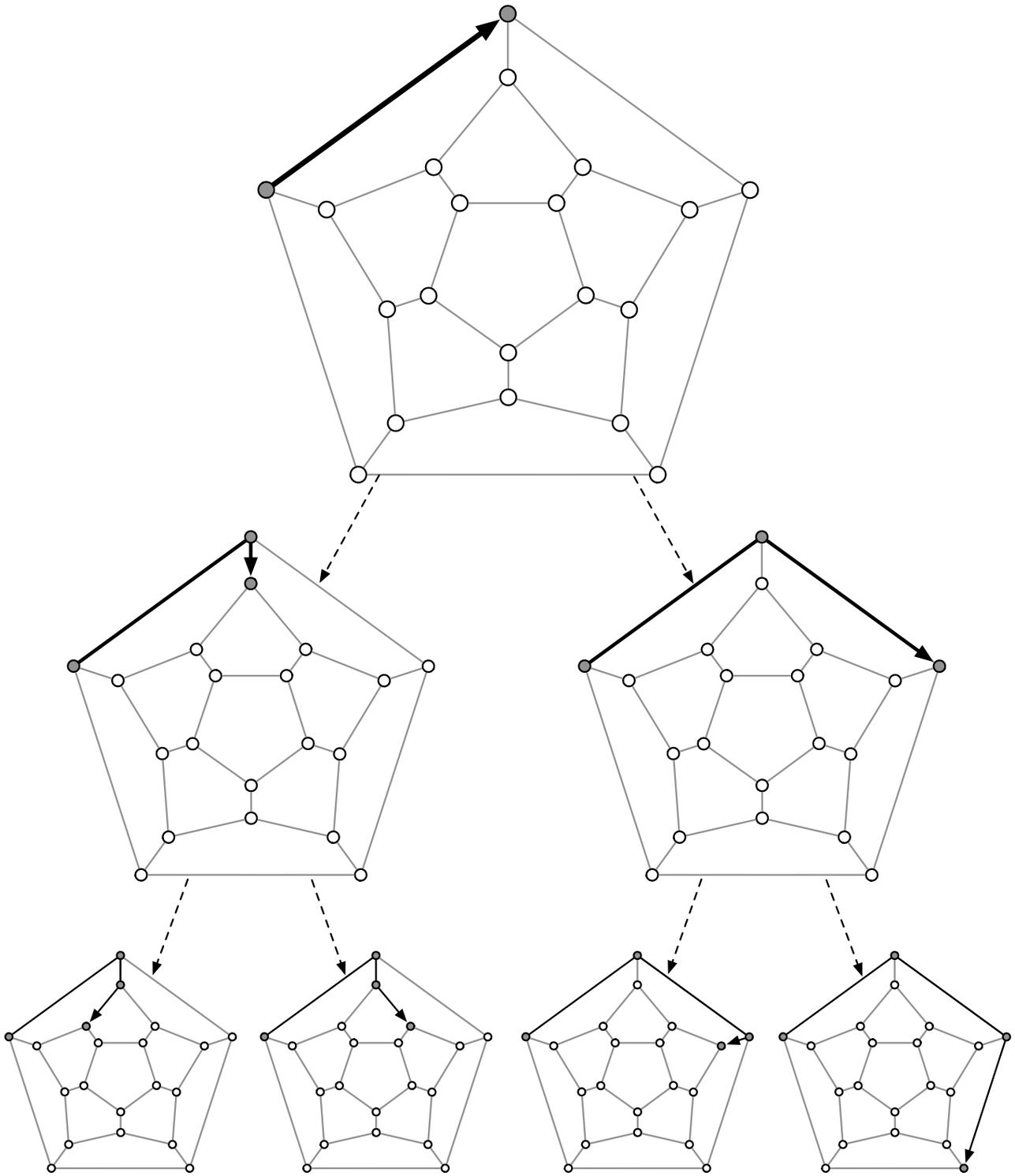
Needles in haystacks

P: we can find a solution efficiently

NP: we can *check* a solution efficiently: tilings, tours, proofs...



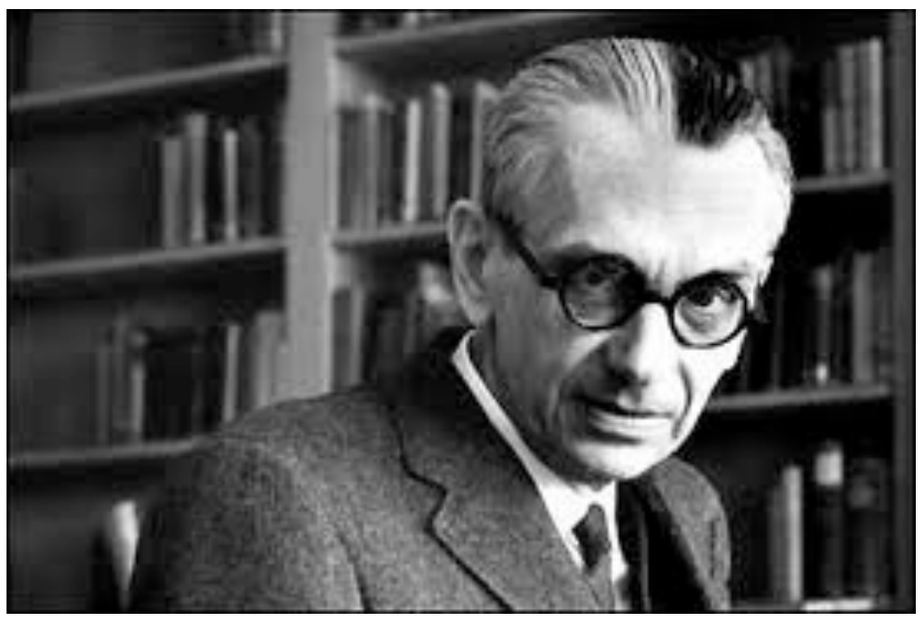
Do we need to search?



P vs. NP

Let $\varphi(n)$ be the time it takes to tell if a proof of length n or less exists…

The question is, how fast does $\varphi(n)$ grow for an optimal machine. One can show that $\varphi(n) \geq Kn$. If there actually were a machine with $\varphi(n) \sim Kn$ (or even only $\varphi(n) \sim Kn^2$), this would have consequences of the greatest magnitude. That is to say, it would clearly indicate that, despite the unsolvability of the *Entscheidungsproblem*, the mental effort of the mathematician in the case of yes-or-no questions could be completely replaced by machines (footnote: apart from the postulation of axioms). One would simply have to select an n large enough that, if the machine yields no result, there would then be no reason to think further about the problem.



The diagonal argument

The problem Halt(Π,x) of telling whether a program Π will halt on input *x* can't be decidable...

since if it were we could run the following program:

```
Trouble(Π)
    if Π(Π) will ever halt, then loop forever
    else halt
```

and `Trouble(Trouble)` would halt if and only if it wouldn't.

But Halt is RE, so

$$\text{DECIDABLE} \subset \text{RE}.$$

The diagonal argument in bounded time

We can't predict whether or not Π(Π) will halt within t steps, with fewer than *t* steps of computation...

since if we could we could run the following program:

$\texttt{Trouble}_t(\Pi)$
  if $\Pi(\Pi)$ will halt in t or fewer steps, then loop forever
  else halt

and $\texttt{Trouble}_t(\texttt{Trouble}_t)$ would halt if and only if it wouldn't.

But a universal program can simulate *t* steps of Π in *s*(*t*) steps, so if *g* ≪ *f*,

$$\mathrm{TIME}(g(n)) \subset \mathrm{TIME}(s(f(n)).$$

TMs: $s(t) = O(t \log t)$
RAMs: $s(t) = O(t)$

The time hierarchy theorem

This proves [Hartmanis and Stearns, 1965]

$$\mathrm{TIME}(n) \subset \mathrm{TIME}(n^2) \subset \mathrm{TIME}(n^{2.001}) \subset \cdots \subset \mathrm{P} \subset \mathrm{EXPTIME} \subset \cdots$$

Similar theorems for SPACE, NTIME, SPACE... can compare apples to apples

But can a similar argument separate P and NP?

We can't seem to diagonalize P within NP — but perhaps some other type of diagonalization will work?

Sadly, no...

Oracles and relativization

We can consult an oracle for a set *A*, asking her yes-or-no questions

P^A is the class of problems we can solve in polynomial time, with her help

NP^A is the class of problems where we can check solutions in polynomial time, with her help

[Baker, Gill, Solovay 1975]: there exist oracles *A*, *B* such that

$$P^A = NP^A \quad \text{but} \quad P^B \neq NP^B$$

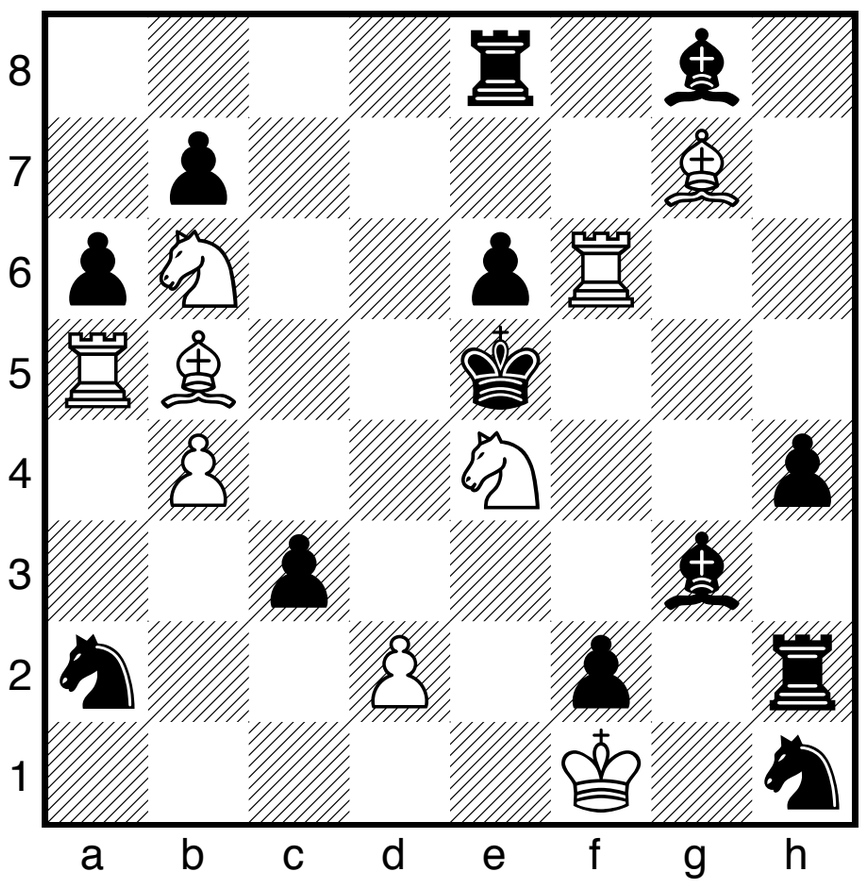
Logical hierarchies

I can win if there exists a move for me,

such that for all of your replies,

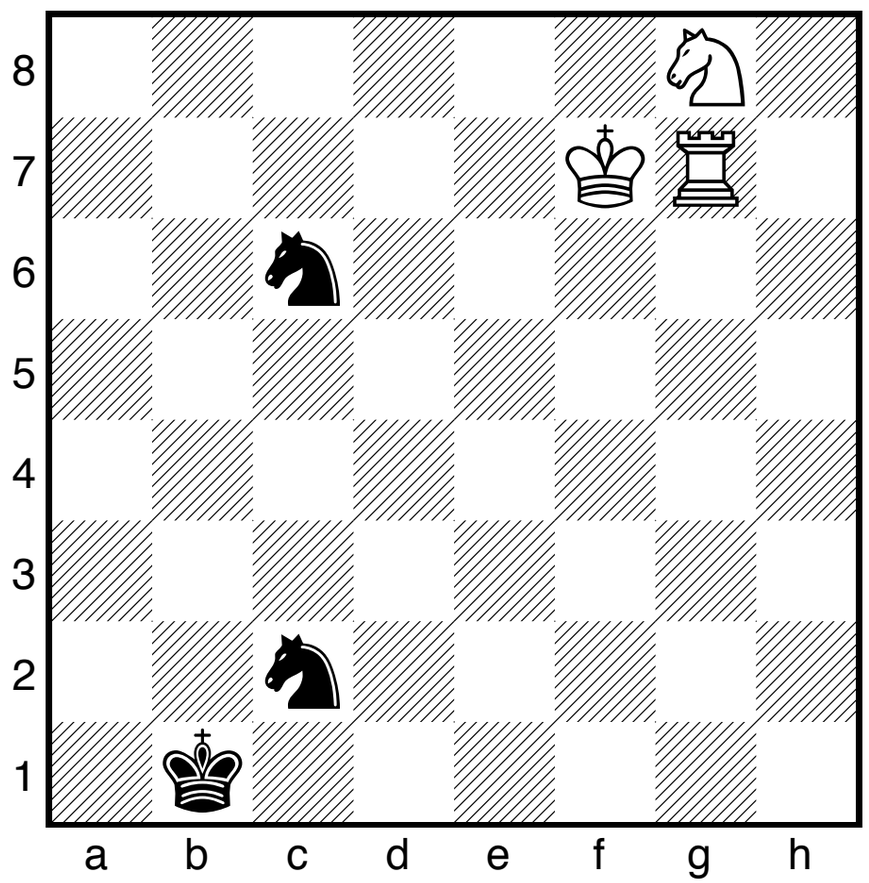
there exists a move for me...

Sam Loyd (1903)



Mate in 3

Lewis Stiller (1995)



Mate in 262

A powerful oracle

Adding poly(n) quantifiers to P gives the class

$$\mathrm{PSPACE} = \exists\forall\exists\cdots\mathrm{P}$$

Let A be a PSPACE-complete problem, such as Quantified SAT:

$$\exists x_1 : \forall y_1 : \exists x_2 : \cdots : \forall y_n : \phi(x_1, y_1, x_2, \ldots, y_n)$$

NP^A is P^A with another $\exists$. This just gives another instance of A, so

$$\mathrm{P}^A = \mathrm{NP}^A .$$

A random oracle

For each n=0, 1, 2, ... flip a coin

If Heads, choose a random string s_n of length n:
The oracle likes s_n, dislikes all others of length n

If Tails, the oracle dislikes all strings of length n

Haystack(n): is there a string of length n that the oracle likes?

In NP^B, since if the answer is “yes” we can prove it by asking her about s_n

But (with probability 1) not in P^B, since we have no hope of finding s_n among the 2^n possibilities in only poly(n) time.

$$\mathrm{P}^B \neq \mathrm{NP}^B .$$

A barrier to resolving the P vs. NP question

Since P vs. NP has different answers in different “possible worlds”...

...no proof technique that *relativizes* (works in the presence of oracles) can resolve it either way

includes diagonalization, and also ideas like exhaustive search:

$$\mathrm{NP} \subseteq \mathrm{EXPTIME}, \quad \mathrm{NTIME}(f(n)) \subseteq \mathrm{TIME}(2^{O(f(n))})$$

“syntactic” manipulations of programs are not enough

Another barrier: natural proofs

We would like to say that a function *f* is outside a complexity class C if *f* is "too complicated"

But if "complicatedness" is

common — i.e. random functions are complicated

constructive — it is fairly easy to compute from *f*'s truth table

then this leads to a contradiction if C contains *pseudorandom* functions...

...and we think P does!

Defeats most known techniques in circuit complexity: random restrictions, Fourier methods, etc. [Razborov and Rudich, 1994]

Further generalized to "algebrization" [Aaronson and Wigderson, 2009]

Phase transitions in NP-complete problems

3-SAT: Boolean variables $x_1, \ldots, x_n$

Constraints $(x_1 \vee \overline{x_3} \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_{17}}) \wedge \cdots$

Is there a truth assignment for $x_1, \ldots, x_n$ that satisfies the entire formula, i.e., all the constraints?

A classic NP-complete problem: any problem in NP can be reduced (translated) to it

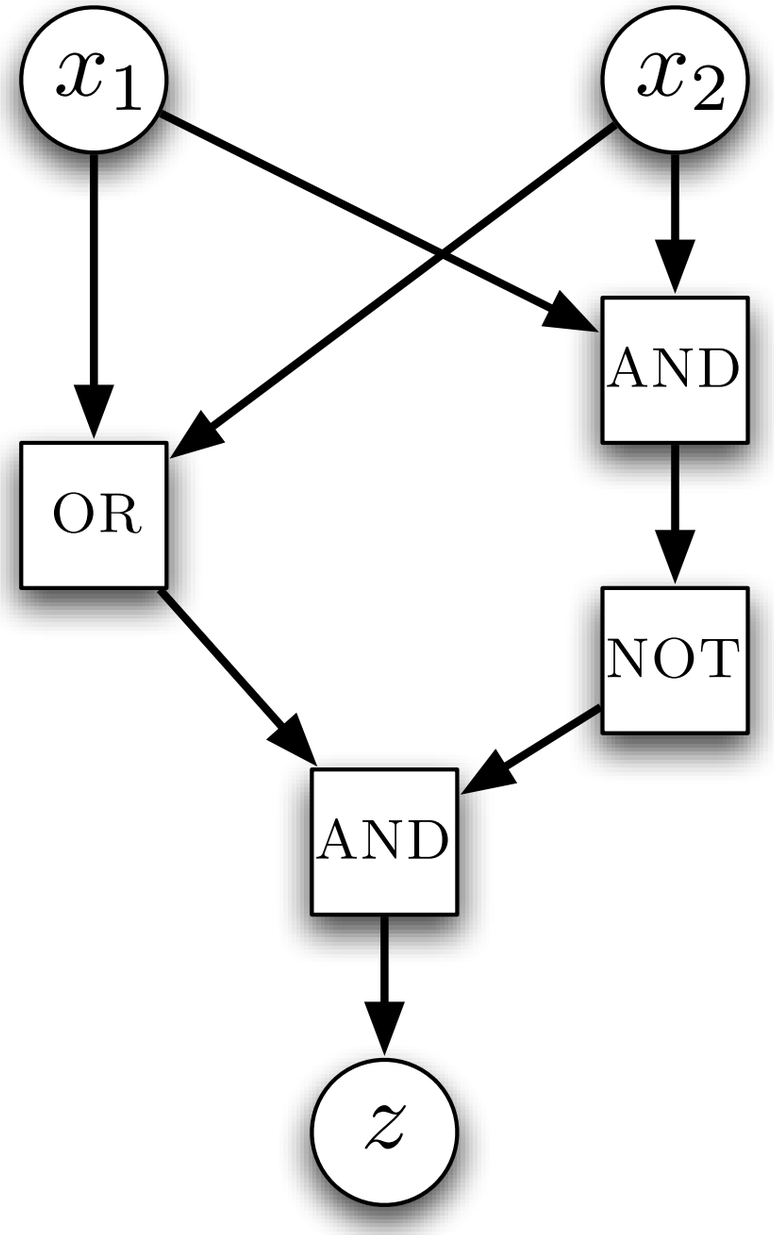
If we can solve it in polynomial time, then P=NP

Satisfying a circuit

Any program that tests solutions (e.g. Hamiltonian paths) can be "compiled" into a Boolean circuit

The circuit outputs "true" if an input solution works

Is there a set of values for the inputs that makes the output true?

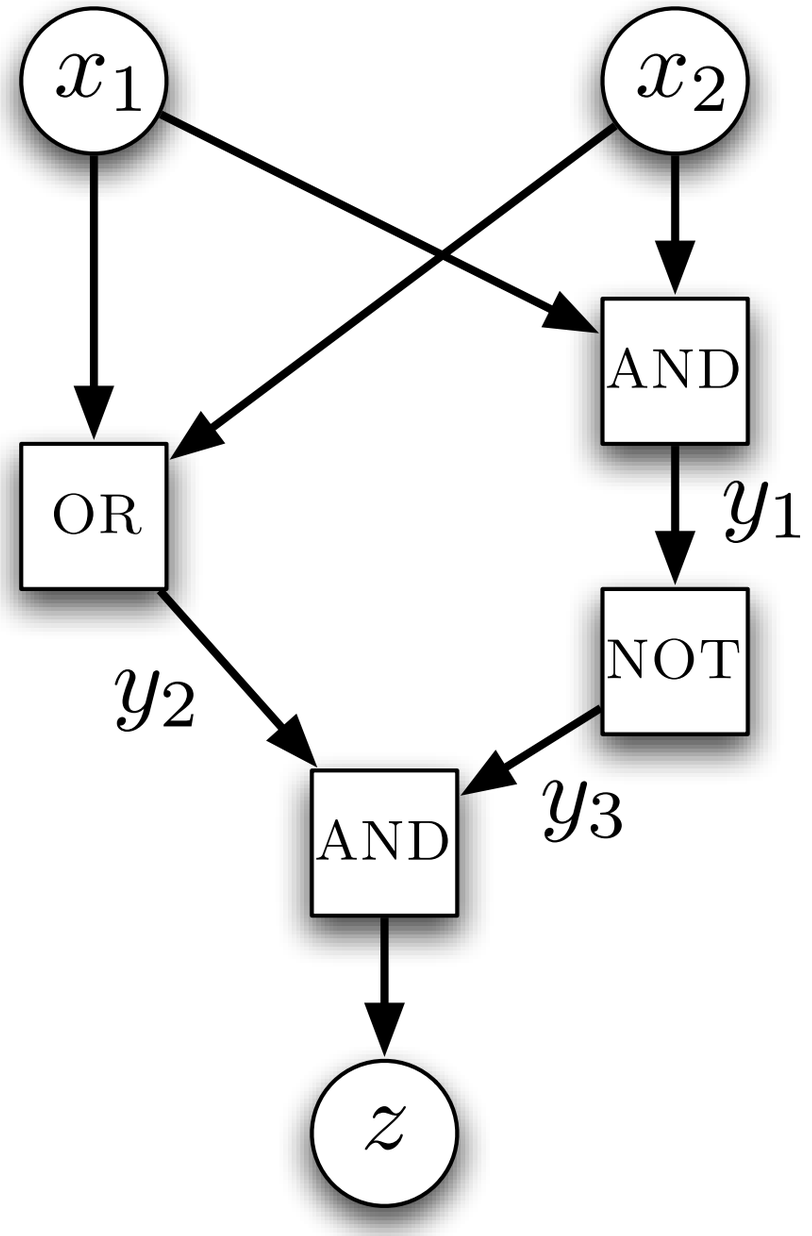


From circuits to formulas

Add variables representing the truth values of the wires

The condition that each AND or OR gate works, and the output is “true,” can be written as a Boolean formula:

$$(x_1 \vee \overline{y}_1) \wedge (x_2 \vee \overline{y}_1) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee y_1) \wedge \cdots \wedge z \ .$$



Phase transitions in NP-complete problems

NP-completeness is a worst-case notion

3-SAT is hard because hard instances exist...

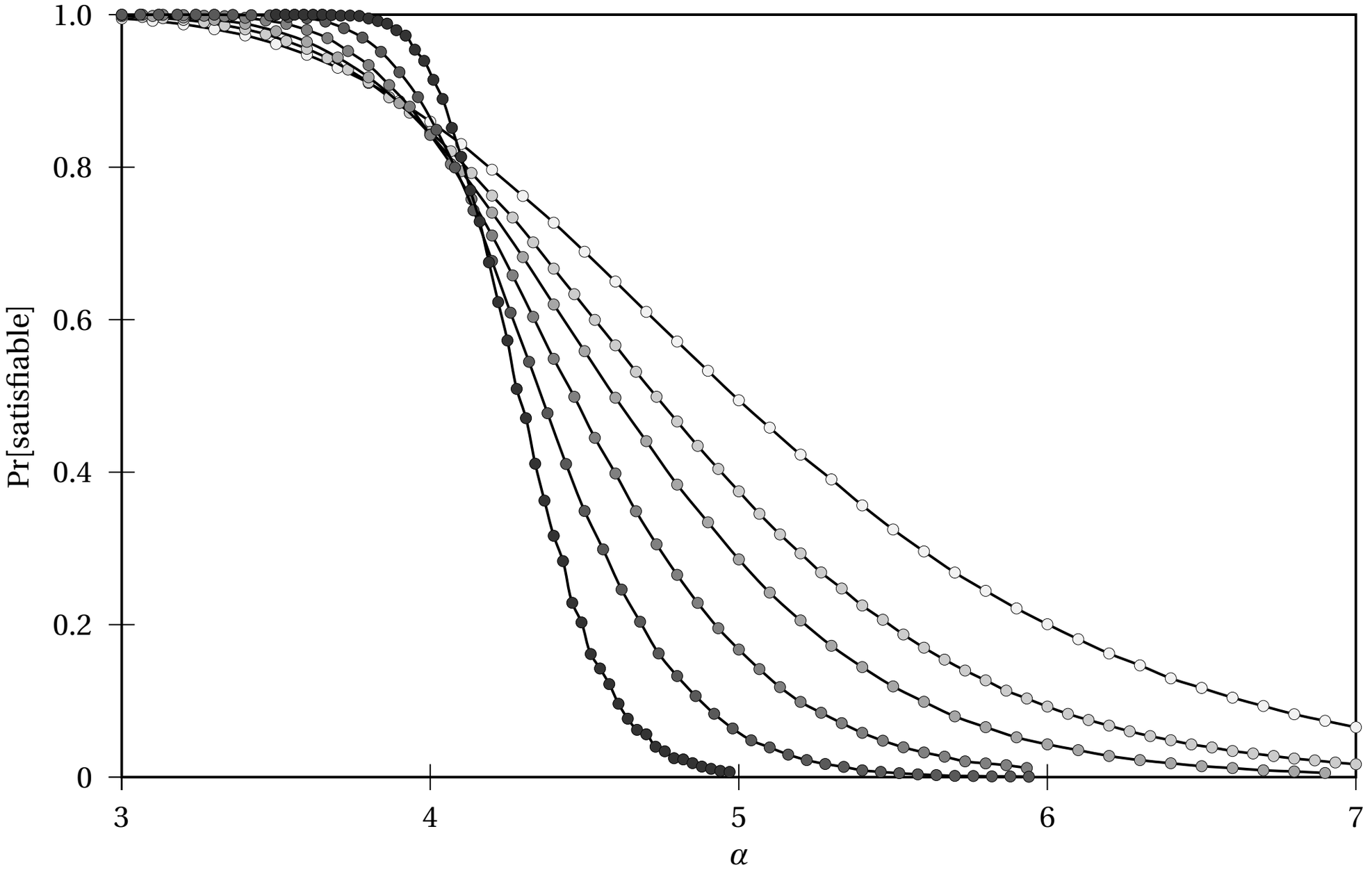
...and we assume instances are designed by a clever adversary (cruel world!)

What if the constraints are chosen randomly instead?

As we add more constraints, more contradictions arise...

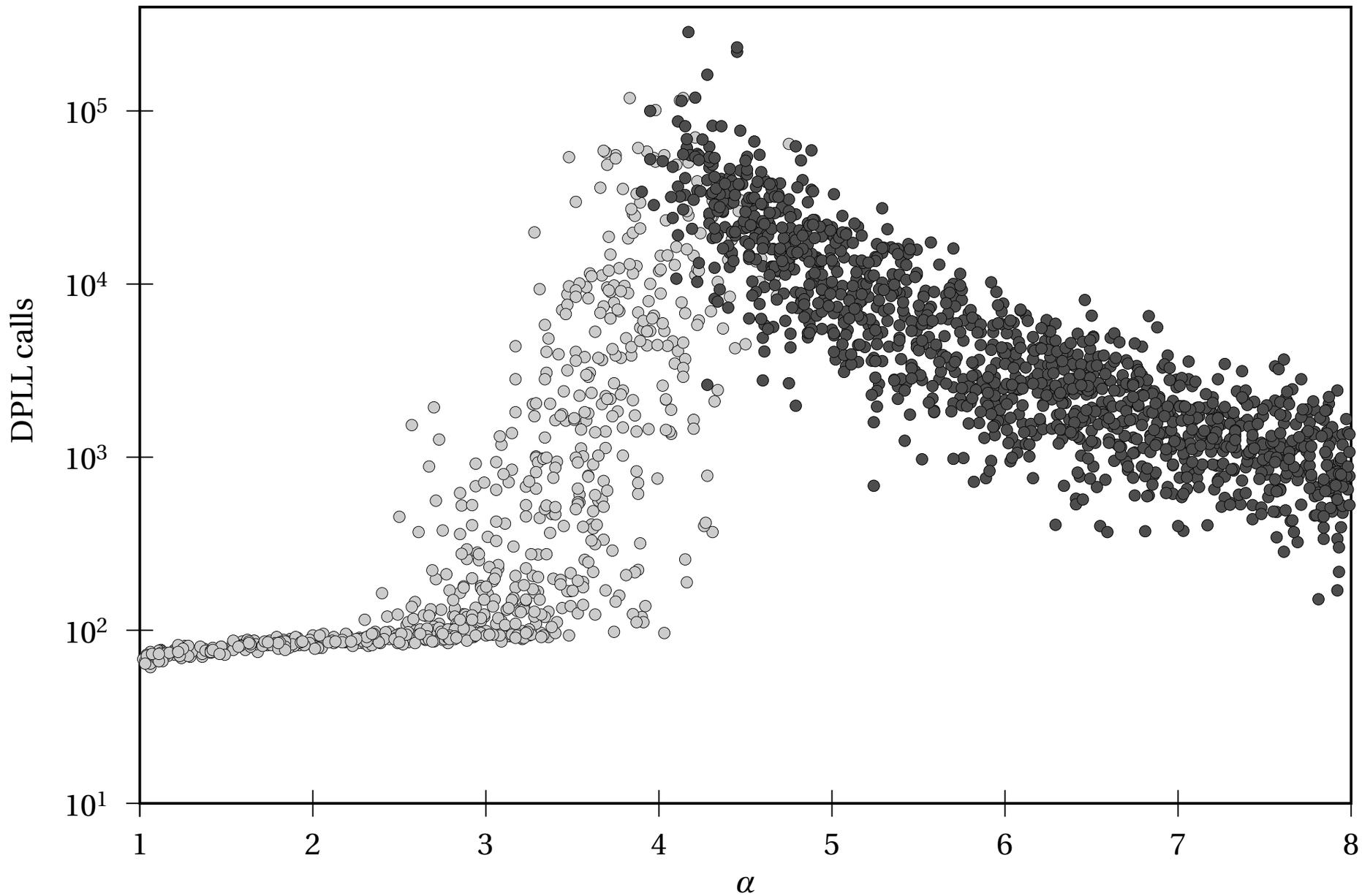
A transition from solvability to unsolvability

When the density of constraints is too high, we can no longer satisfy all of them

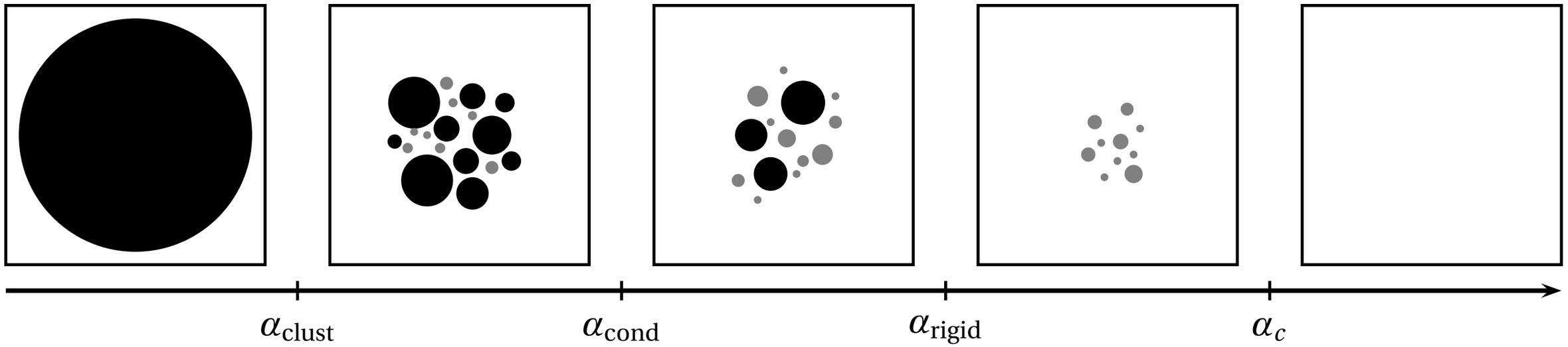


Where the hard problems are

Search times are highest at the boundary



Clustering, freezing, and hardness



At a certain density, solutions break up into clusters

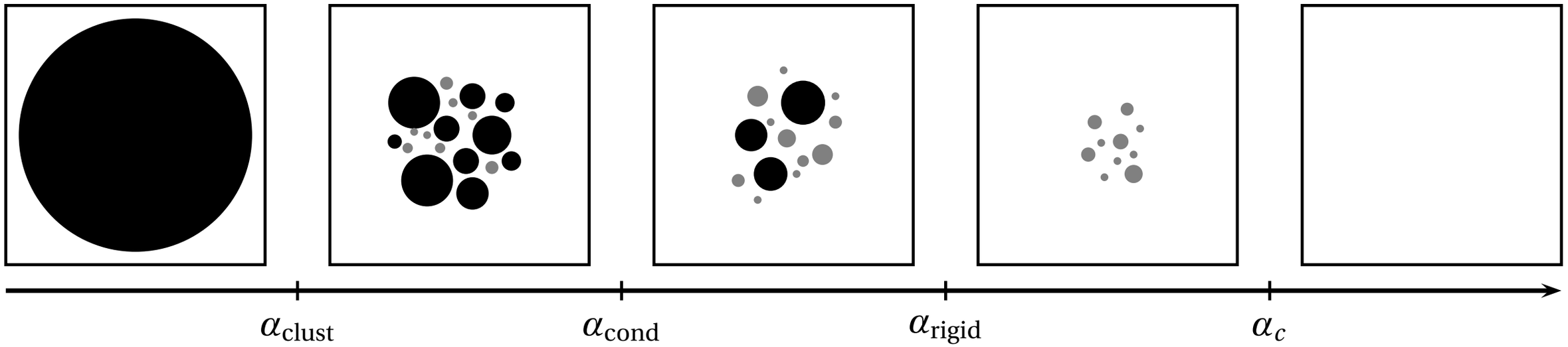
These clusters become *frozen* — many variables take a fixed value

If a search algorithm sets any of these variables wrong, it's doomed

A rugged landscape, with many local optima to get stuck in

[Achlioptas, Coja-Oghlan, Krzakala, Mezard, Molloy, Montanari, Moore, Ricci-Tersenghi, Zdeborová , Zecchina...]

Clustering, freezing, and hardness



We believe that the “freezing” transition marks where the problem becomes hard

All known algorithms for *k*-SAT stop working at or below $\alpha_{\text{rigid}} \sim 2^k \log k / k$

Hard, but satisfiable, instances up to $\alpha_c \sim 2^k \log 2$

Can this be made into a proof that P≠NP?

XORSAT

Use XOR (addition mod 2) instead of OR:

$$x_1 \oplus x_2 \oplus x_3 = 1$$
$$x_1 \oplus x_2 \oplus x_4 = 0$$
$$x_2 \oplus x_3 \oplus x_4 = 1$$

Random instances have many of the same properties as 3-SAT: clustering and freezing (at the same density) and then a transition to unsatisfiability

But XORSAT is easy! Just linear equations:

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

XORSAT

How is XORSAT like SAT, and how is it different?

Clustering: local search algorithms can’t move through the space, and hill-climbing algorithms get stuck in local optima

Freezing: backtracking algorithms (Davis-Putnam) take exponential time, repeatedly setting frozen variables the wrong way

But Gaussian elimination is a global rearrangement of variables and constraints, letting us turn a hard-looking problem into an easy one

If SAT has a similar kind of rearrangement — something totally different from backtracking or local search — then P=NP

Proving that it doesn’t is hard!

The road ahead

It seems unlikely that P vs. NP is formally independent since it is almost first-order:

Consider the statement “For all $n \geq 1000$, there is no circuit of size $n^{\log n}$ that solves all SAT formulas of length n” [Ben-David & Halevi, Aaronson]

Sophisticated approaches from algebraic geometry have been suggested [Mulmuley]

The most hopeful view: we will eventually prove that P≠NP...

...but we will be forced to build a lot of new mathematics in the process!

Problems in the gap

To prove that a problem *A* is undecidable, we usually reduce Halting to it:

$$\textsc{Halting} \leq A$$

But there are undecidable problems to which Halting can't be reduced: "easier" than (or at least different from) than Halting [Friedberg-Muchnik]

If P≠NP, there are problems that are outside P, but not NP-complete [Ladner]

Candidates: Factoring, Graph Isomorphism, Shortest Lattice Vector

Could "naturally occurring" problems live in this middle ground?

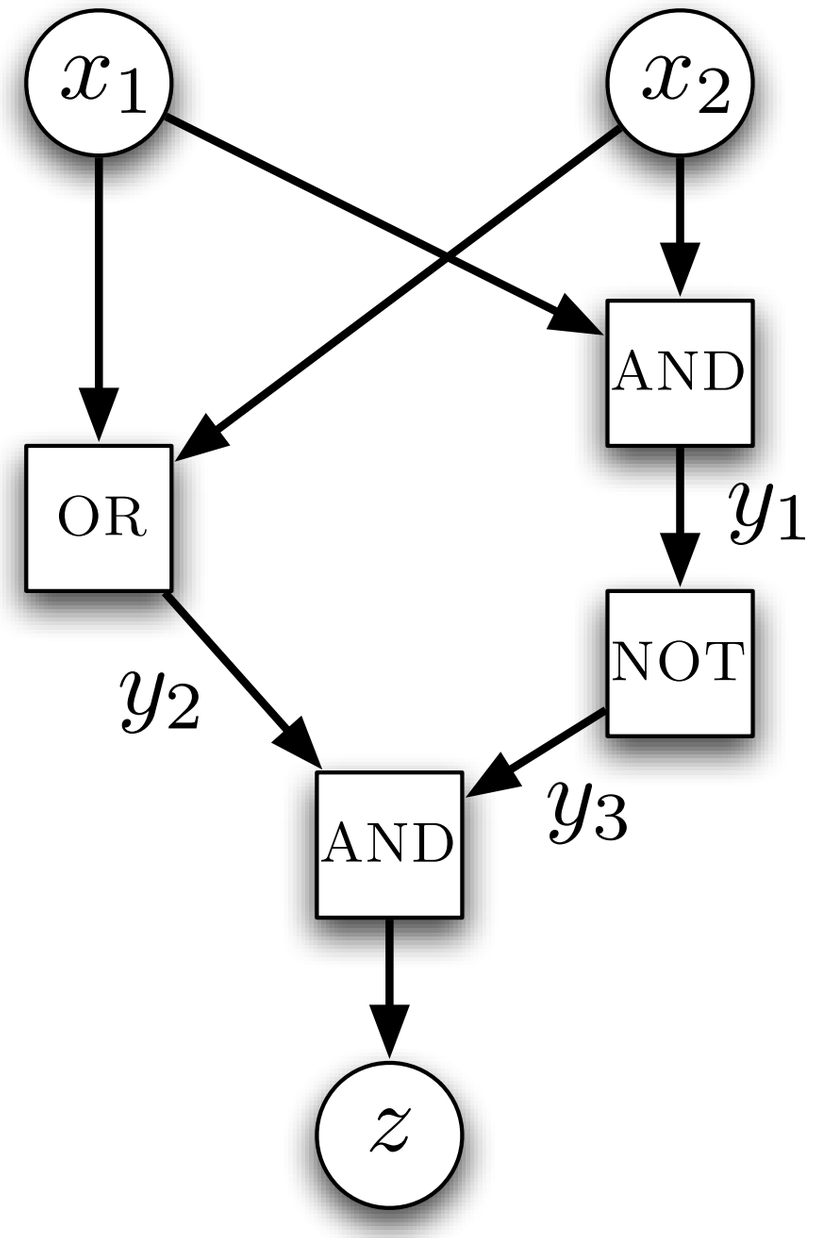
Claim: problems equivalent to Halting are easy (to think about, not to solve!)

Building a computer: SAT clauses

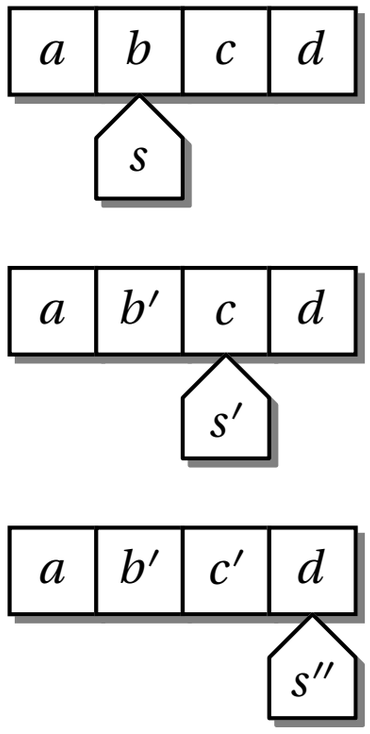
Add variables representing the truth values of the wires

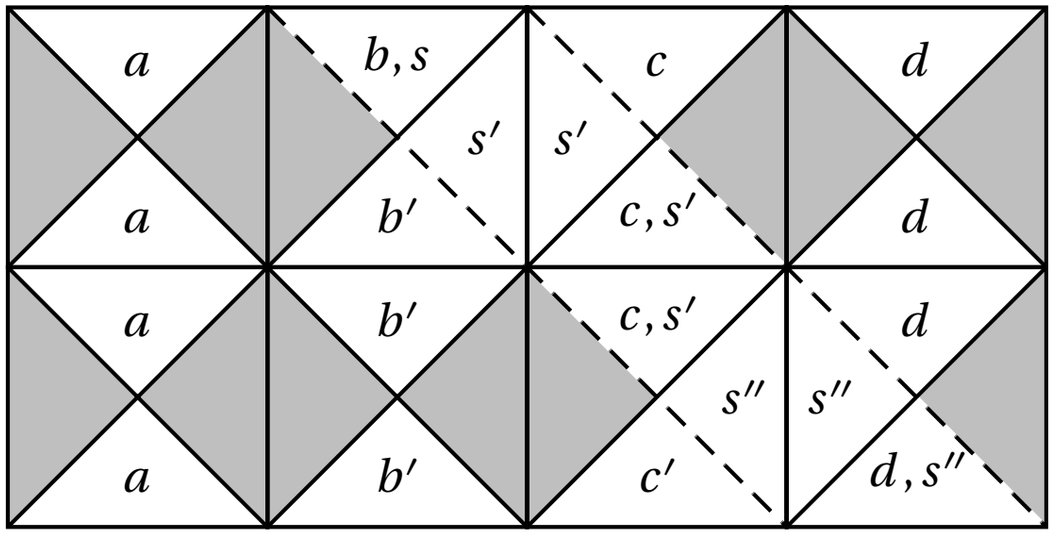
The condition that each AND or OR gate works, and the output is “true,” can be written as a Boolean formula:

$$(x_1 \vee \overline{y}_1) \wedge (x_2 \vee \overline{y}_1) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee y_1) \wedge \cdots \wedge z \ .$$

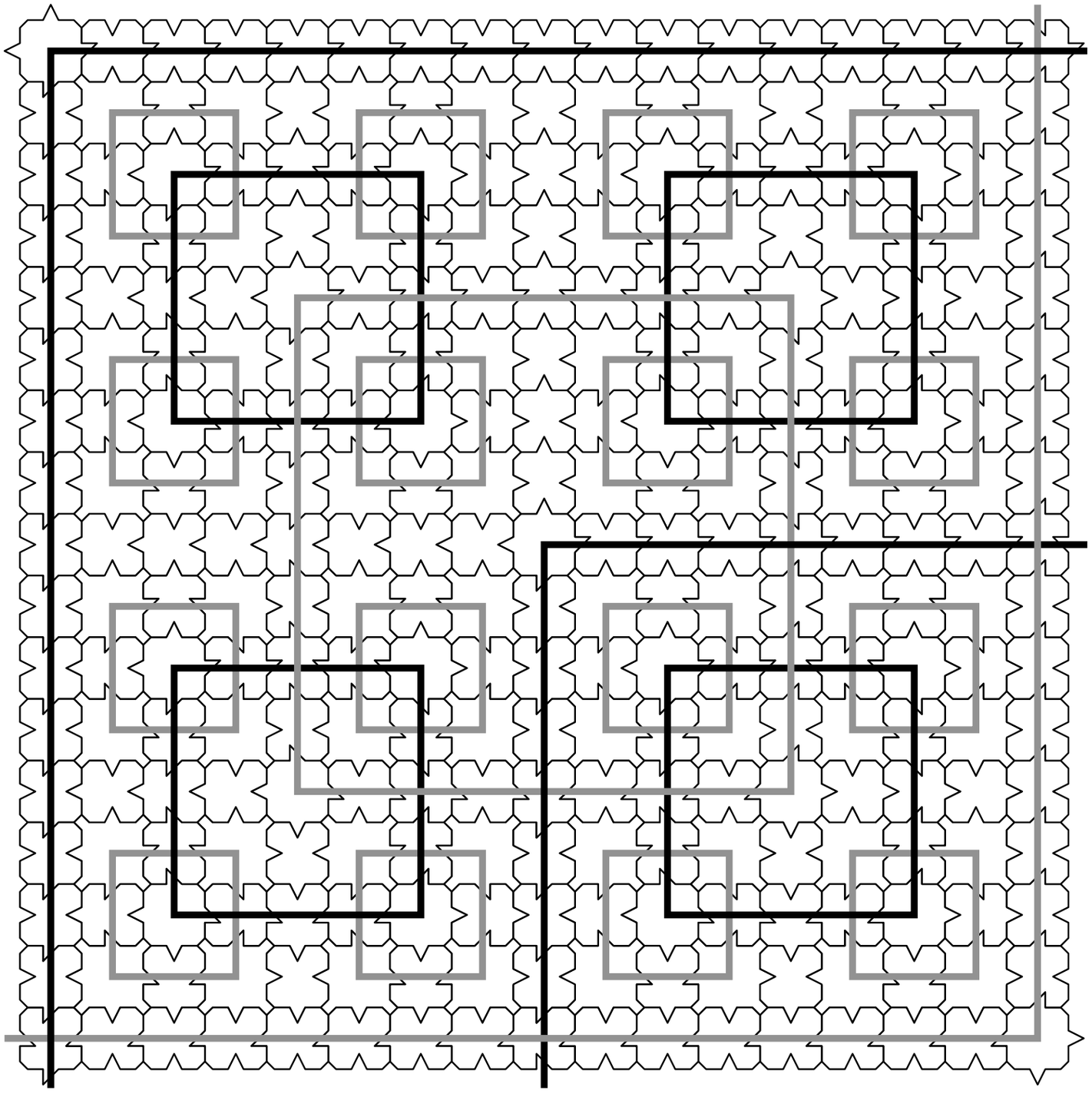


Building a computer: tilings



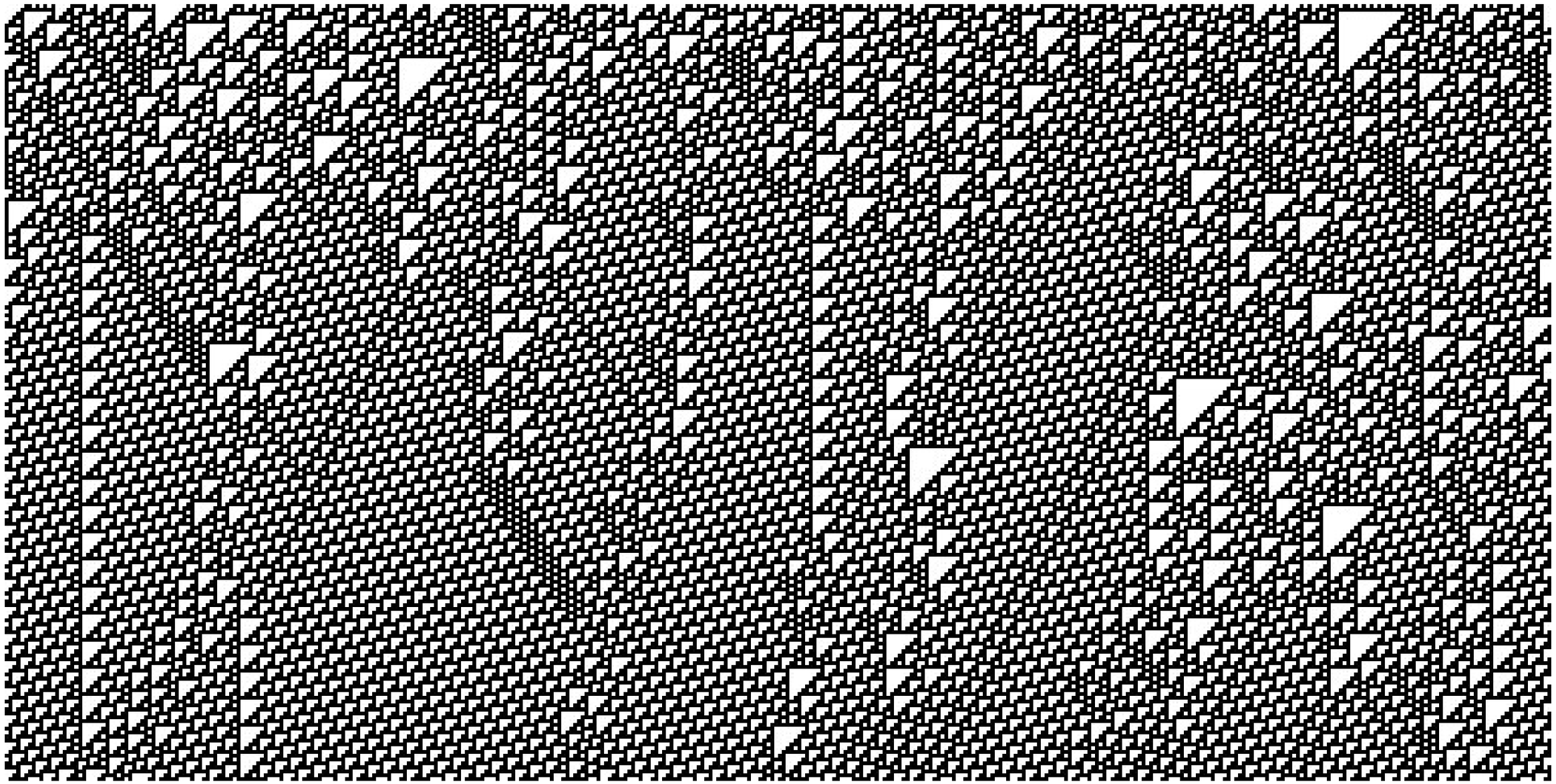


Building a computer: tilings



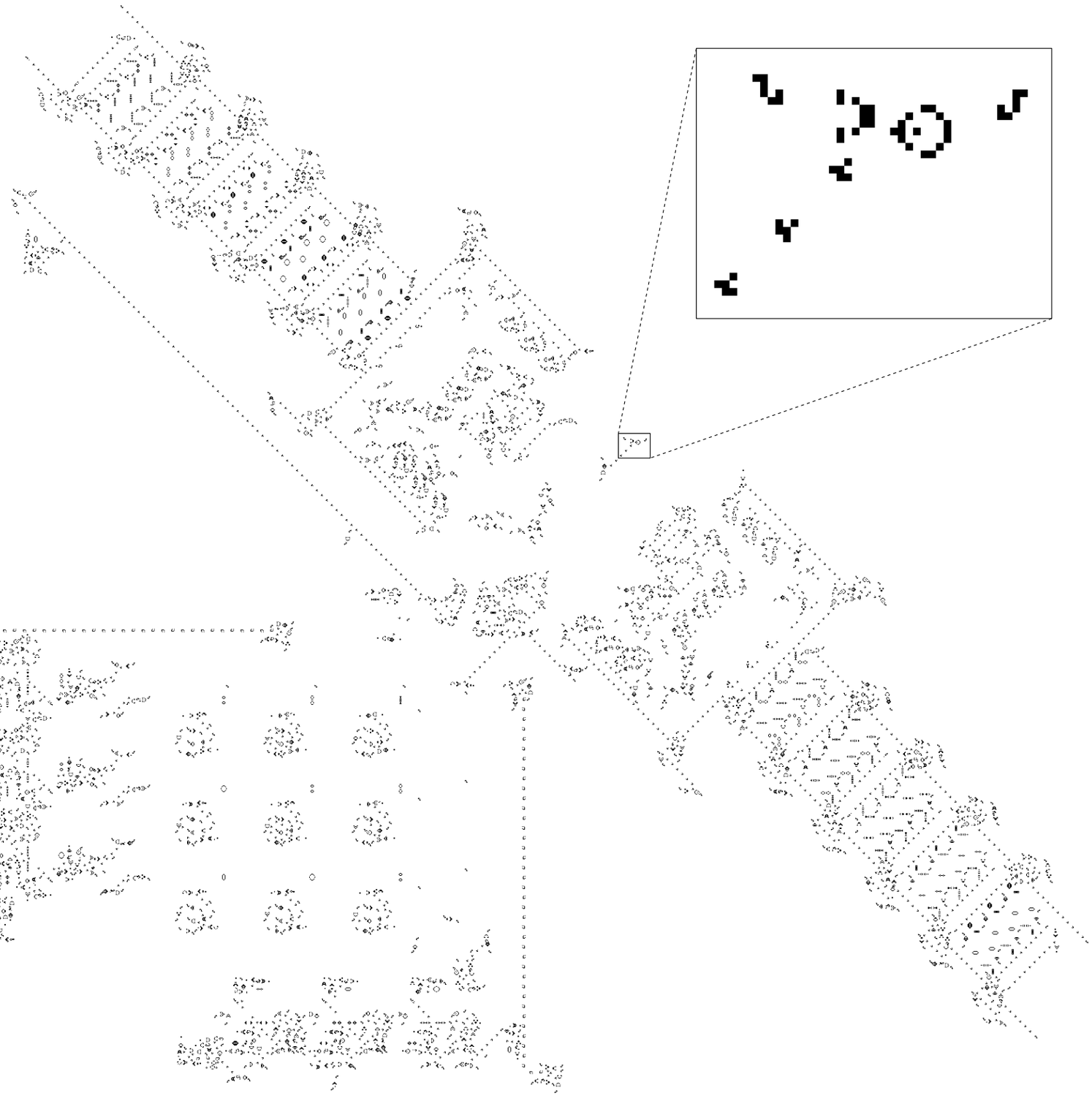
[Robinson]

Building a computer: cellular automata



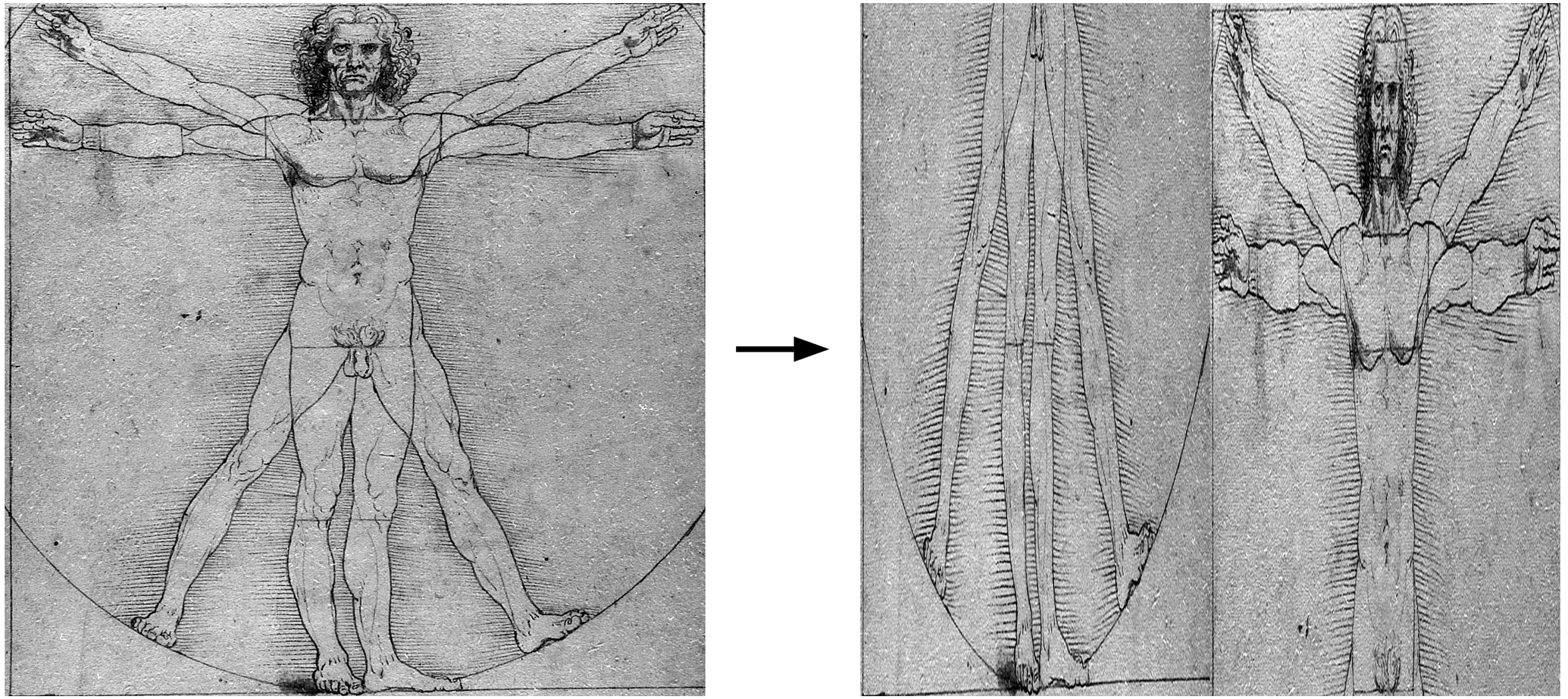
[Cook, Wolfram]

Building a computer: cellular automata



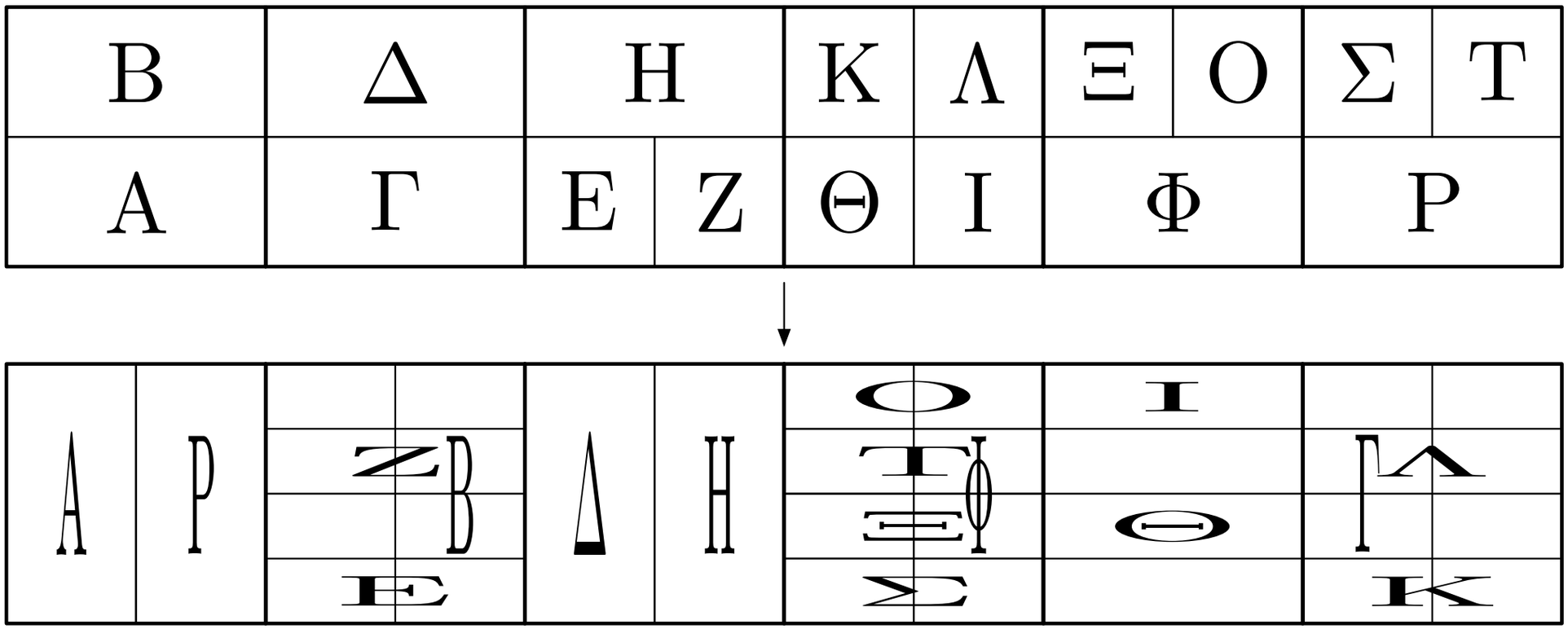
[Rendell]

Building a computer: dynamical systems



$$\ldots y_3 y_2 y_1 \,.\, x_1 x_2 x_3 \ldots$$
$$\Downarrow$$
$$\ldots y_3 y_2 \,.\, y_1 x_1 x_2 x_3 \ldots$$

Building a computer: dynamical systems



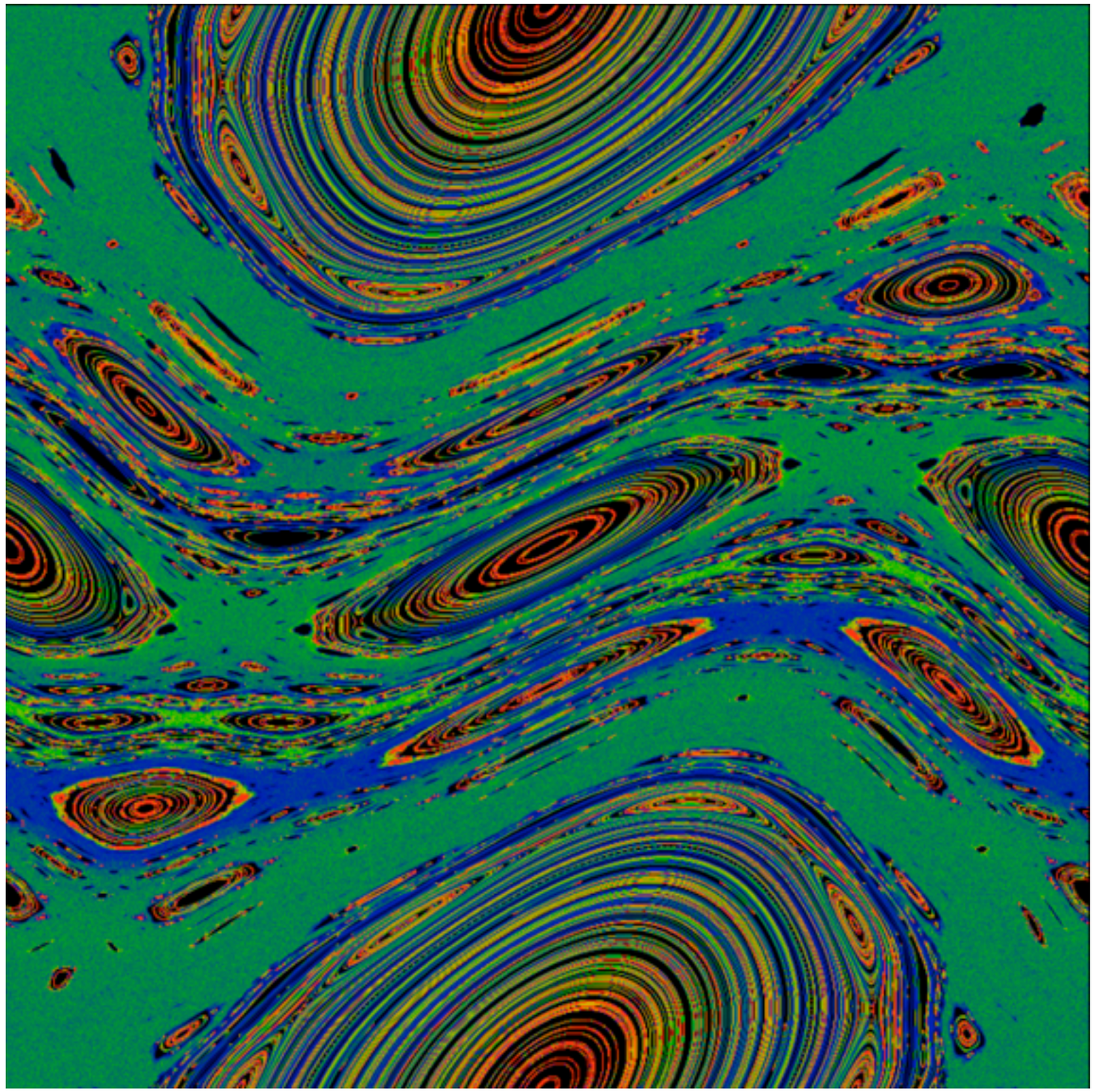
F	s_1	s_2	s_3	s_4	s_5	s_6
0	$0, s_1, L$	$0, s_6, L$	$0, s_2, R$	$1, s_5, R$	$1, s_4, L$	$1, s_1, L$
1	$1, s_2, L$	$0, s_3, L$	$1, s_3, L$	$0, s_6, R$	$1, s_4, R$	$0, s_4, R$

[Moore]

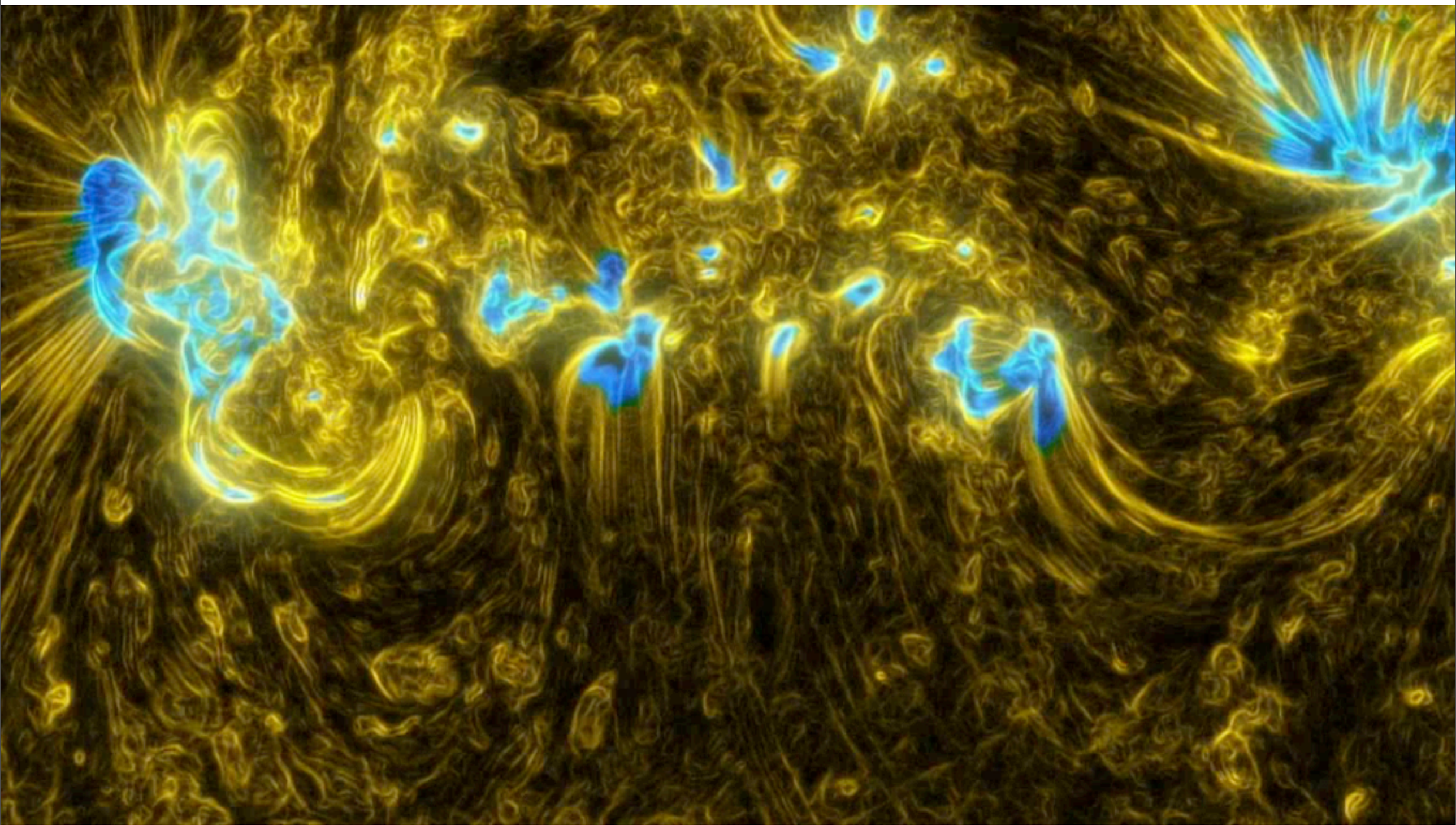
Wild problems

$$p_{t+1} = p_t + K \sin \theta_t$$

$$\theta_{t+1} = \theta_t + p_{t+1}$$



Wild problems



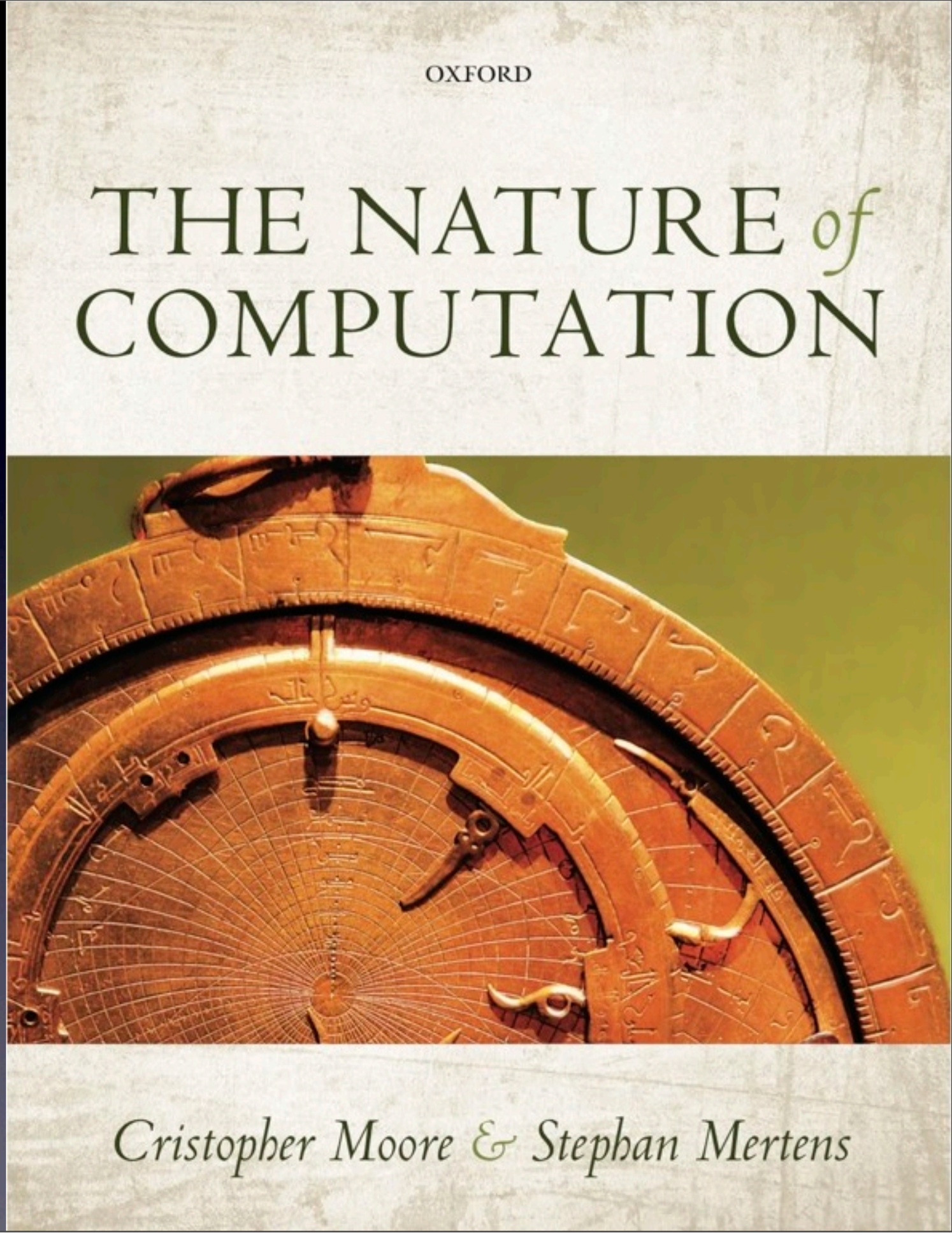
Shameless Plug

This book rocks! You somehow manage to combine the fun of a popular book with the intellectual heft of a textbook.
— Scott Aaronson

A treasure trove of information on algorithms and complexity, presented in the most delightful way.
— Vijay Vazirani

A creative, insightful, and accessible introduction to the theory of computing, written with a keen eye toward the frontiers of the field and a vivid enthusiasm for the subject matter.
— Jon Kleinberg

Oxford University Press, 2011



Acknowledgments

and the National Science Foundation