LECTURE 4
CELLULAR AUTOMATA AND CONWAY'S GAME OF LIFE

CCST9048 Simplifying Complexity
University of Hong Kong
Dr. Tim Wotherspoon

News

- **Tutorial 3 will be optional for all students due to Mid-Autumn Festival Holiday on Monday and Chinese National Day on Thursday.**
  - **Optional means that you get points for participating in Tutorial 3 but others will be treated as an excused absence, even if you are enrolled in a Tuesday section.**
  - **Students from Monday/Thursday sections may attend any Tuesday section if they are available. Space is limited so we will turn away late students if the classroom is overcrowded.**

Group Project and Presentation

Complexity Science has been applied to many different disciplines. Write a report and do a group presentation on a concrete application of complexity to one of the areas listed below.

- Economics/Finance
- Biology
- Ecology
- Medicine
- Politics
- Arts/Architecture
- Engineering
- Sport
- Transportation/Logistics
- Information Technology

Proposal Due 4:00pm on 23 October

Photograph Portfolio

Through a series of photographs, you must tell a story of an instance of complexity in your community, explain why this is such an instance and attempt to identify some of the simple mechanisms that create the emergent property without presence of a designer.

Some possible themes may be:

- Human Interactions
- Architecture and Urban Growth
- Wildlife and Swarms

DUE 5:00pm 13 November

Course Learning Objectives

1. Identify complexity in the global society in multiple fields ranging from biology to physics.
2. Utilize the methods of complexity theory to propose possible solutions to unsolved problems.
3. Explain the key differences between systems based approaches and reductionism.
4. Evaluate how complexity is shaping the interaction between humanity and the global environment.

Definition

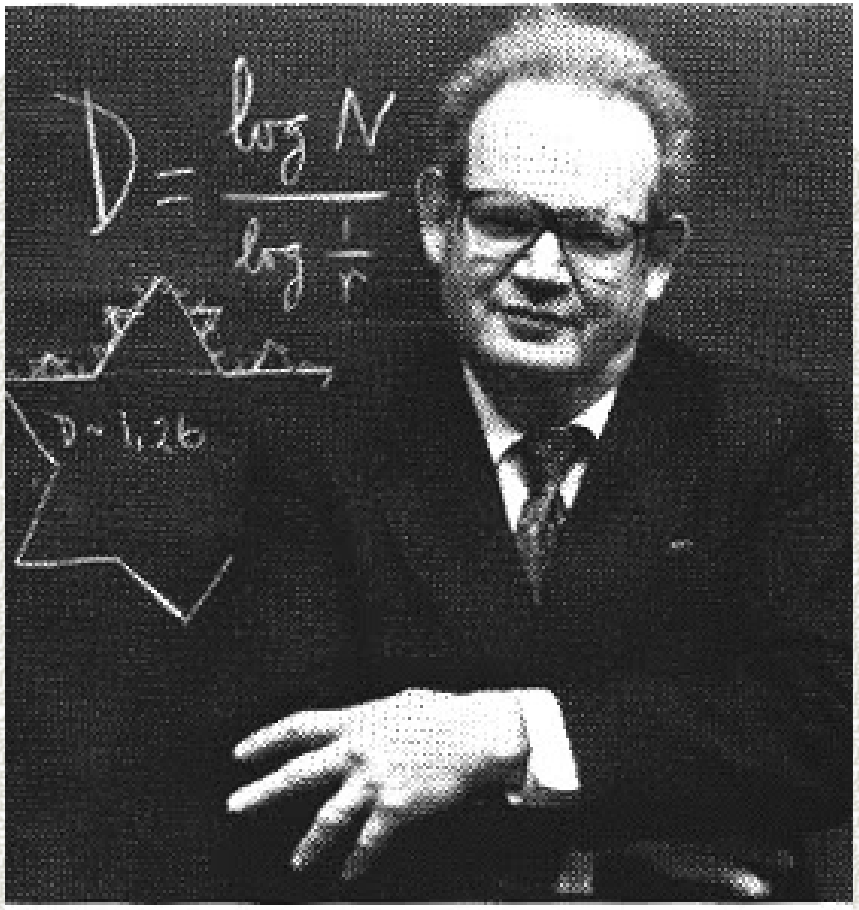
“Complex system: a system in which large networks of components with no central control and simple rules of operation give rise to complex collective behavior, sophisticated information processing and adaptation via learning or evolution.”

Mitchell 2011

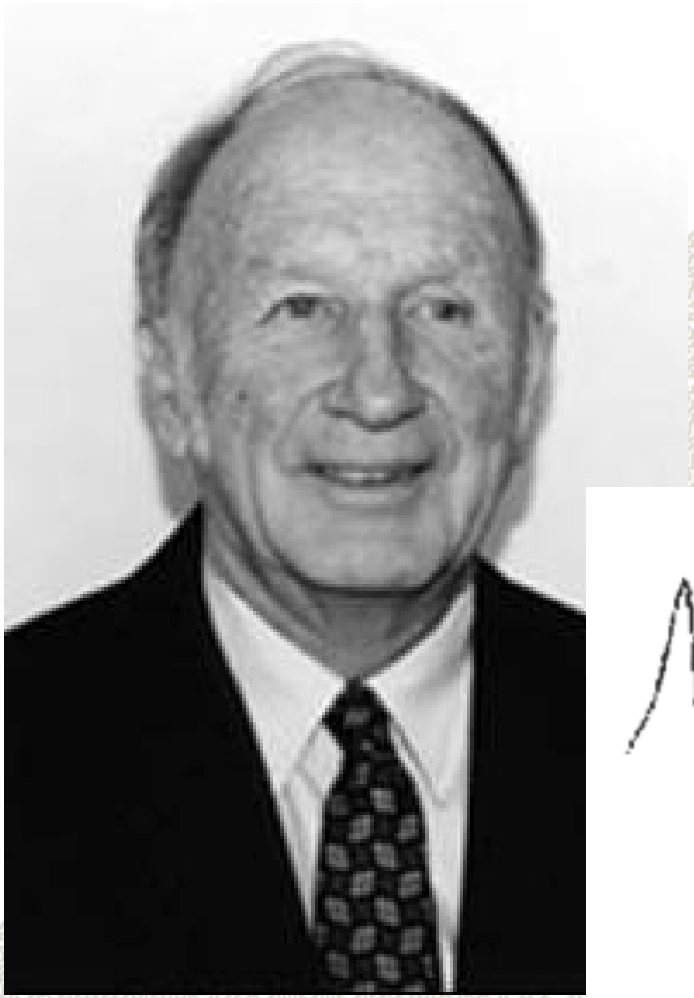
Julia and Mandelbrot Sets

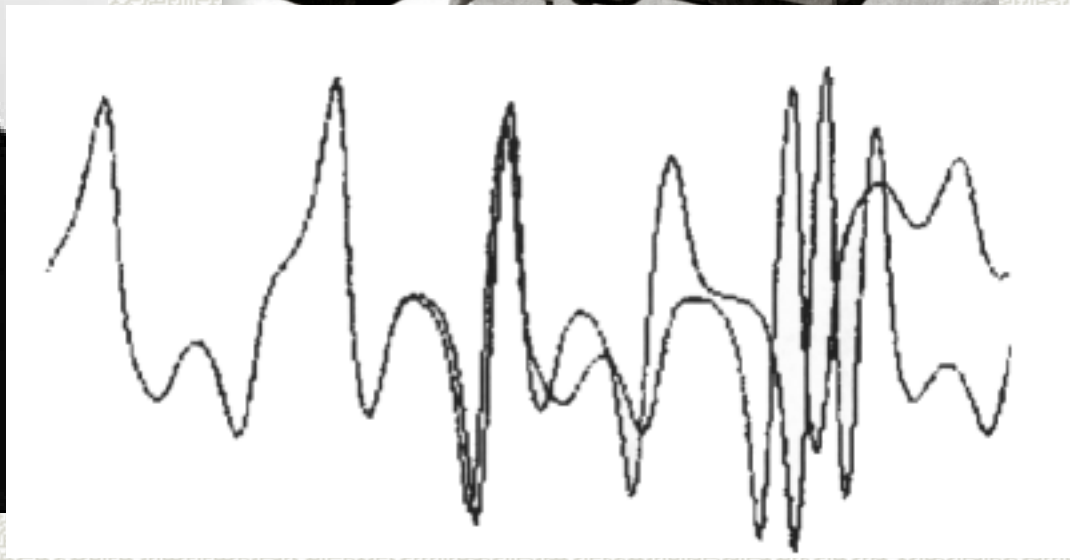
“Bottomless wonders spring from simple rules which are repeated without end”

TED2010 [Fractals and the art of roughness]

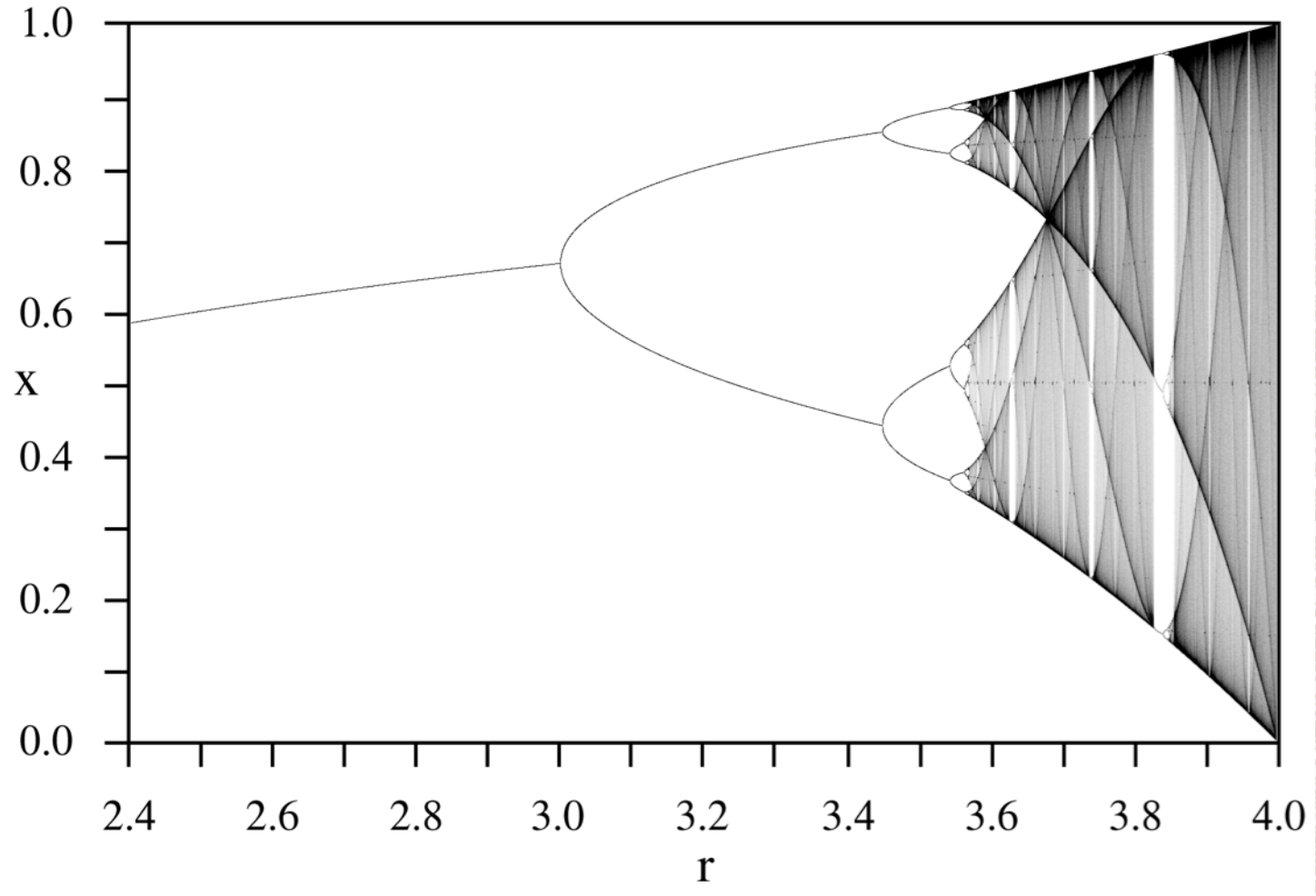


Brief History of Dynamics





Bifurcation Diagram



Defining Chaos

- A function $f:X \to X$ is *chaotic* if and only if f satisfies the following three conditions.
  - f is *transitive*
  - The periodic points of f are *dense* on X.
  - f has *sensitive dependence on initial conditions.*

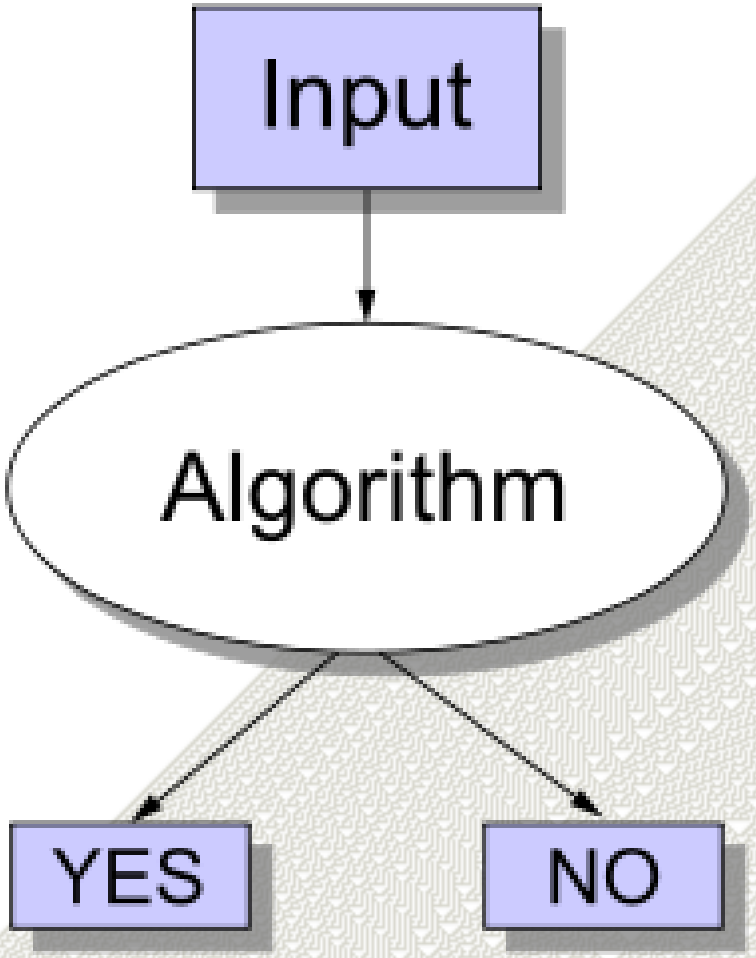
Outline

- Universal Computation
- Conway’s Game of Life
- Elementary Cellular Automata
- Compuational Equivalence

Kahoot

- Chaos Review

Universal Computation



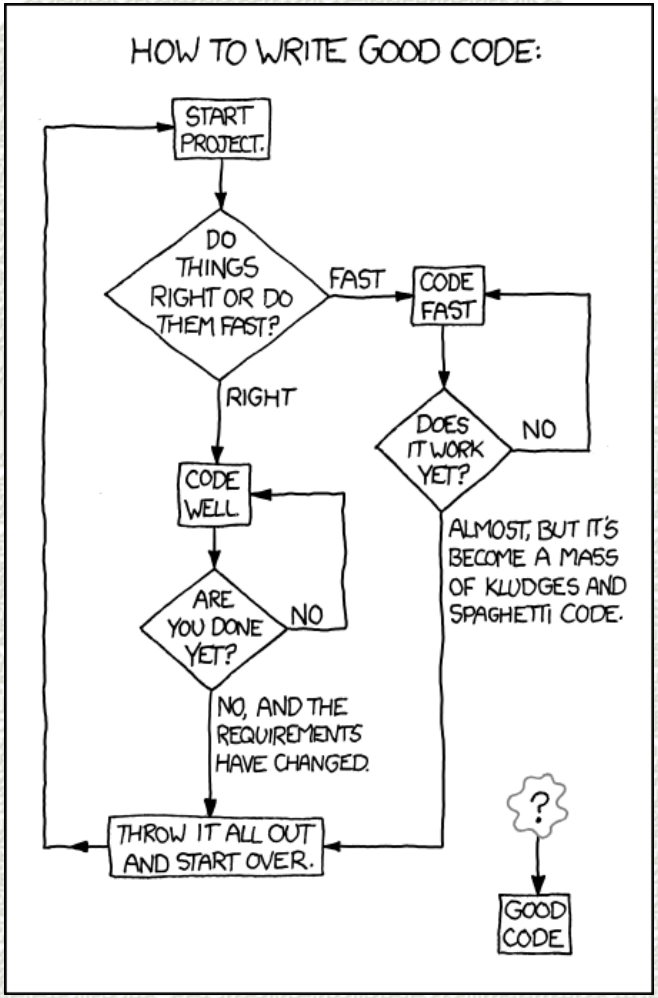
- Is every statement in mathematics decidable?
- Is there a definite procedure that can be applied to every statement that will tell us in finite time whether or not the statement is true or false?

Universal Computation

- Alan Turing attempted this question.
- Very famous for his efficiency at breaking German codes during WWII
- Considered the Father of Computer Science

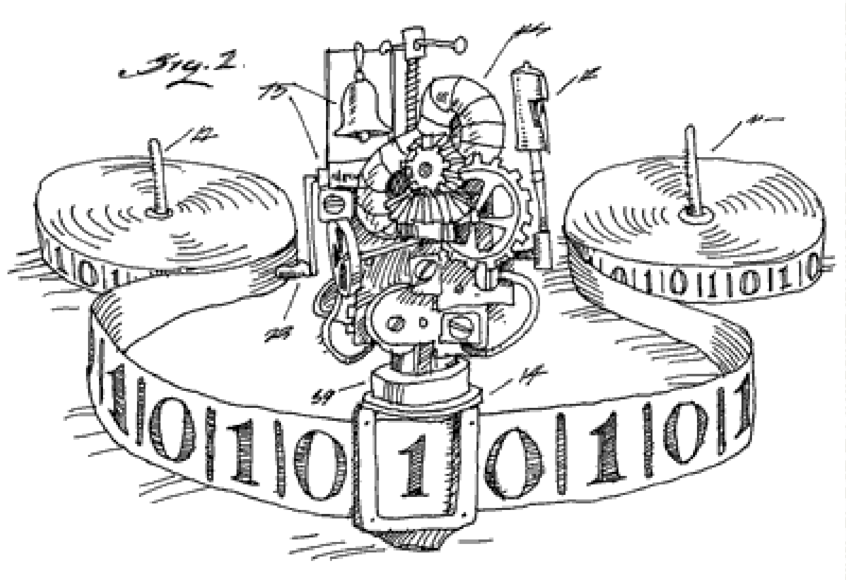
Universal Computation

- Many topics familiar with us were virtually unknown at the time.
- In order to answer this question of whether things are computable, Turing formalized the notion of “definite procedure”.



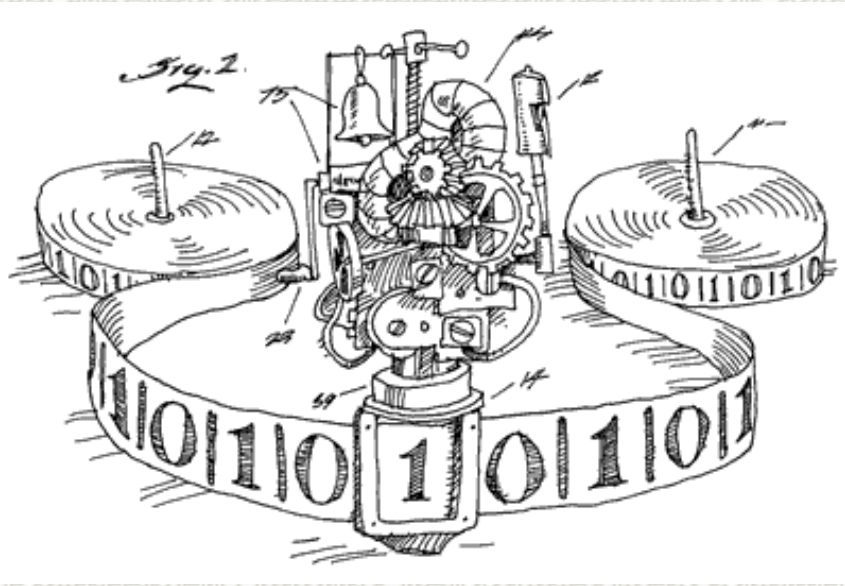
Universal Computation

- This notion is now called a Turing Machine
- The details are not too important for us as to how they work.
- Essentially they were the blueprint of the modern digital computer.



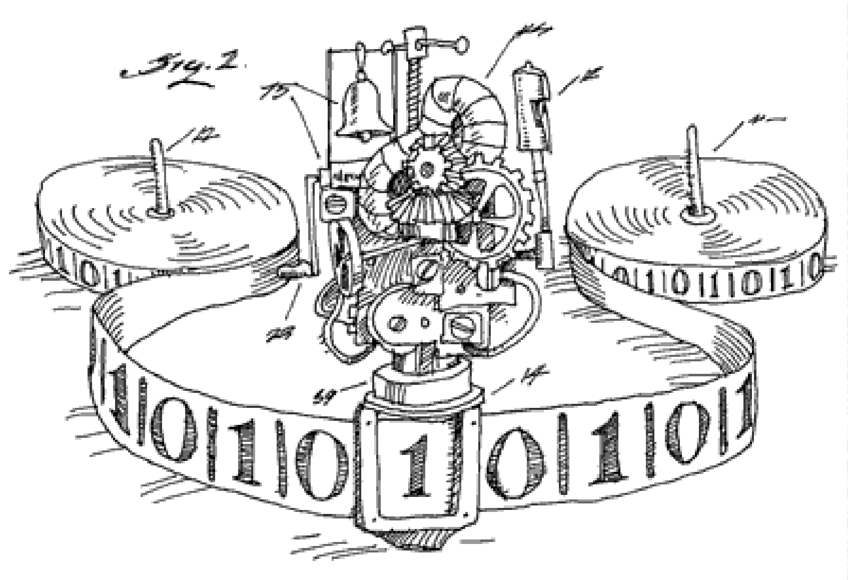
Universal Computation

- Turing machines have a program M and an input I. Turing did this in a very fundamental way with 0s and 1s but we need not think of it so abstractly.
- We can think of the program as a script that it runs on the input.
- Some programs will eventually stop running. That is they will reach a “halt” state. Others may have an infinite loop!



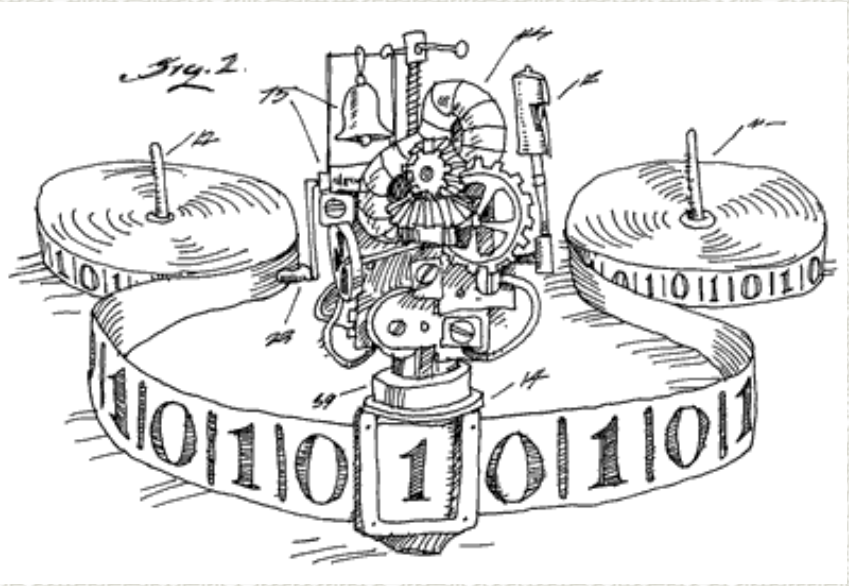
Universal Computation

- Turing was coding both the program and the input as sequences of 0s and 1s so he noticed that it was possible to run the sequence of the program as the input.
- He noted that it was possible to make a "Universal Turing Machine" which was capable of simulating any other Turing Machine



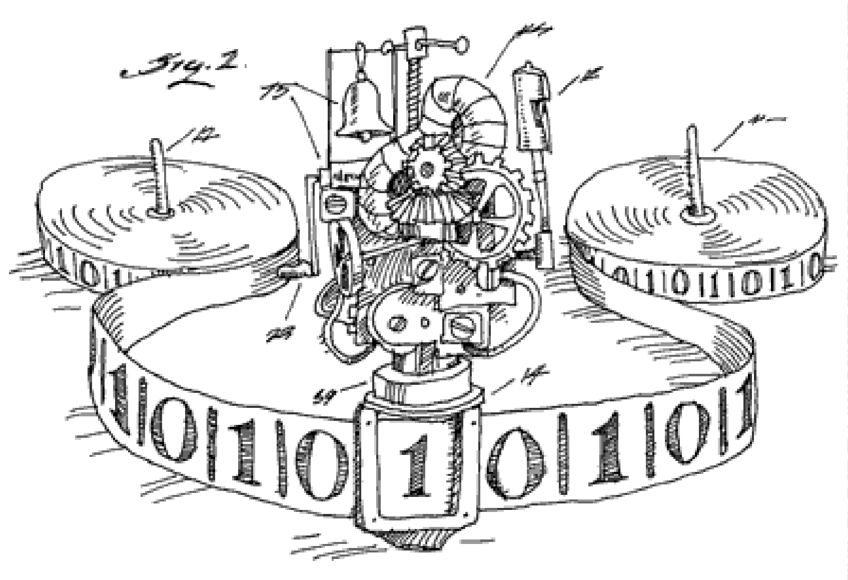
Universal Computation

- Running the program as the input would be like using Microsoft Word to run “Word Count” on the source code of Microsoft Word
- In principle this is easy for us to understand but was very clever in Turing’s day.



Universal Computation

- Turing Machines defined the notion of "definite procedure".
- He then assumed the answer to the question we asked before was "yes".
- He assumes for any statement, there exists a Turing Machine that can decide whether it's true.



Universal Computation

- Begin with the statement “Turing Machine, M, will halt on input, I. “
- Turing machine, H which will decide whether the above statement is true or not.
- By assumption, H exists.

- The input of H should be M and I.
- If M halts on I then H will halt and the output will be YES.
- If M does not halt on I then H will halt and the output will be NO.

Universal Computation

- Finally we make H’ that works exactly the same way as H.
- We will only care about programs that are running themselves as inputs.
- H’ halts only if the answer is NO, M does not halt on it’s own program
- If the answer is YES, M halts on it’s own program, then H’ goes into an infinite loop.
- What does H’ do when it runs on it’s own program?

Universal Computation

- Suppose that H’ halts on the input H’.
- This means that the answer is NO, H’ does not halt on its own program.
- However, it does halt! A contradiction.

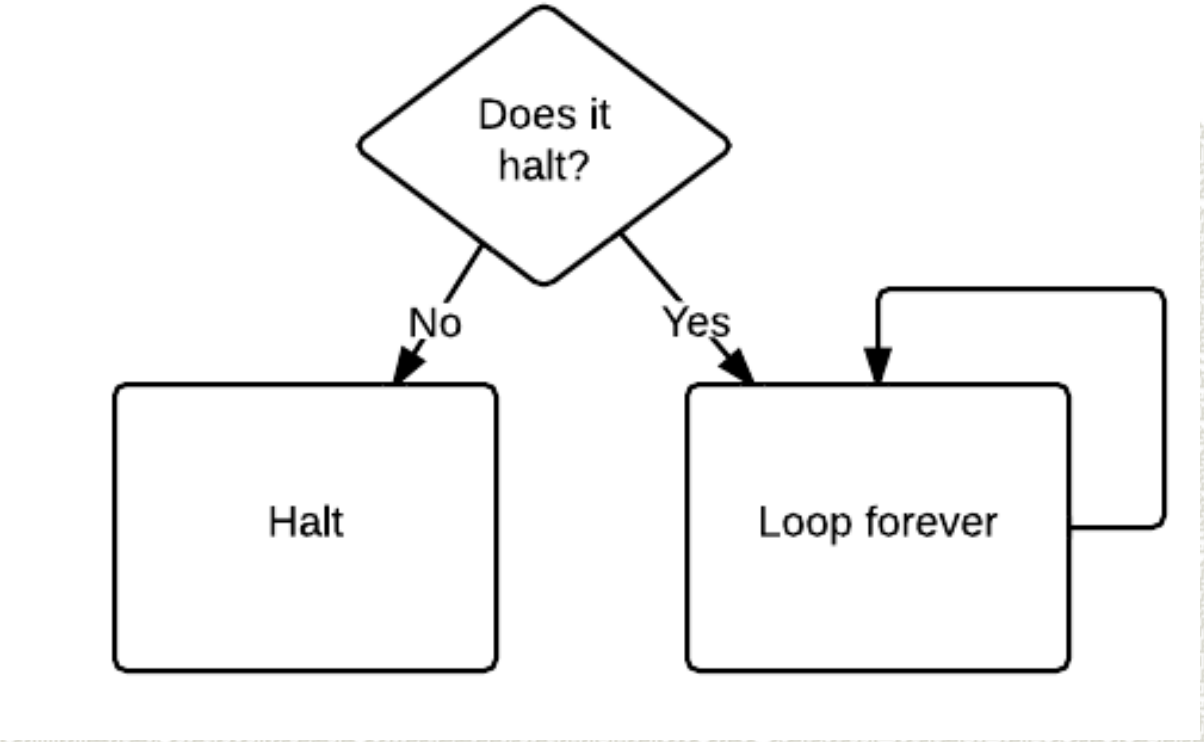
- Suppose now that H’ does not halt on the input H’.
- This means the answer is YES, H’ halts on it’s own program
- But it didn’t!
- Again a contradiction.

Universal Computation

This flow chart outlines the argument.

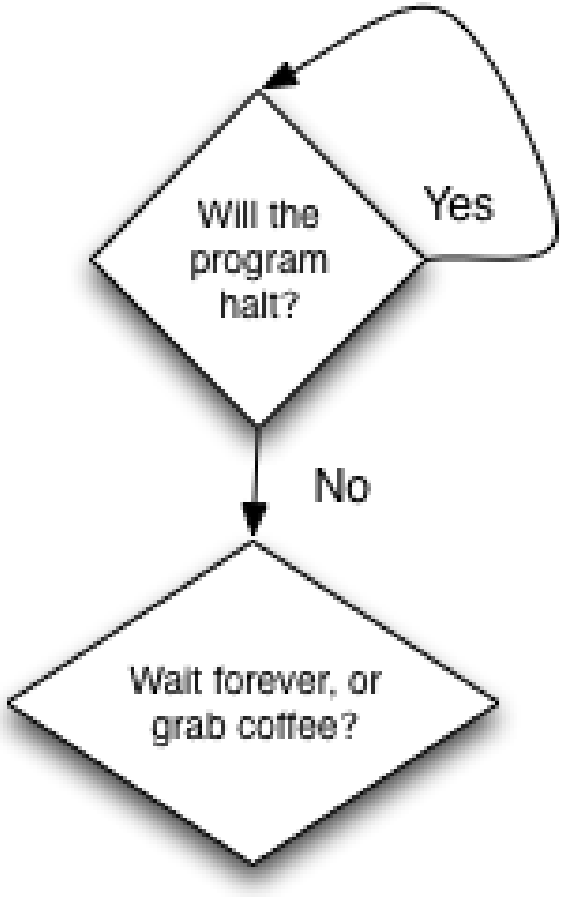
The box Does it halt? Should be thought off as a complicated computer program that can detect infinite loops in other programs.

Can it detect an infinite loop in itself?



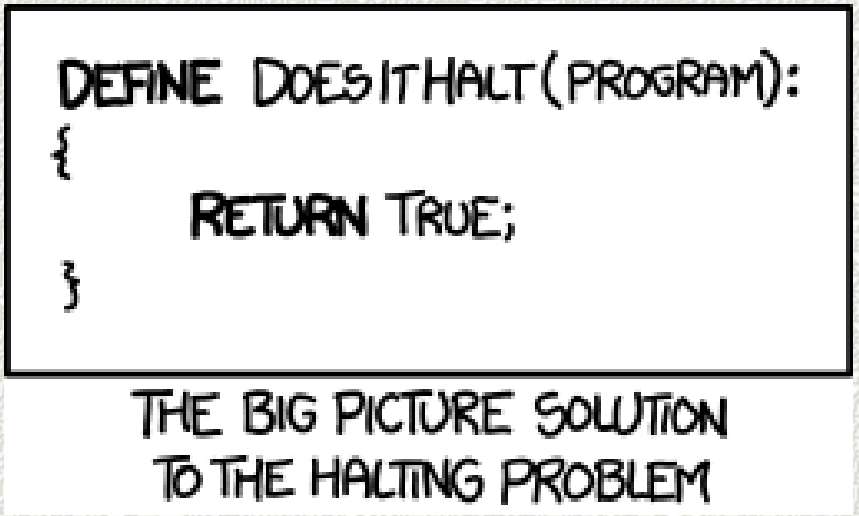
Universal Computation

- H’ cannot exist.
- H’ operates exactly like H. H itself cannot exist!
- There can be no machine that can tell you whether or not Turing Machine M running I will end in a finite number of steps.



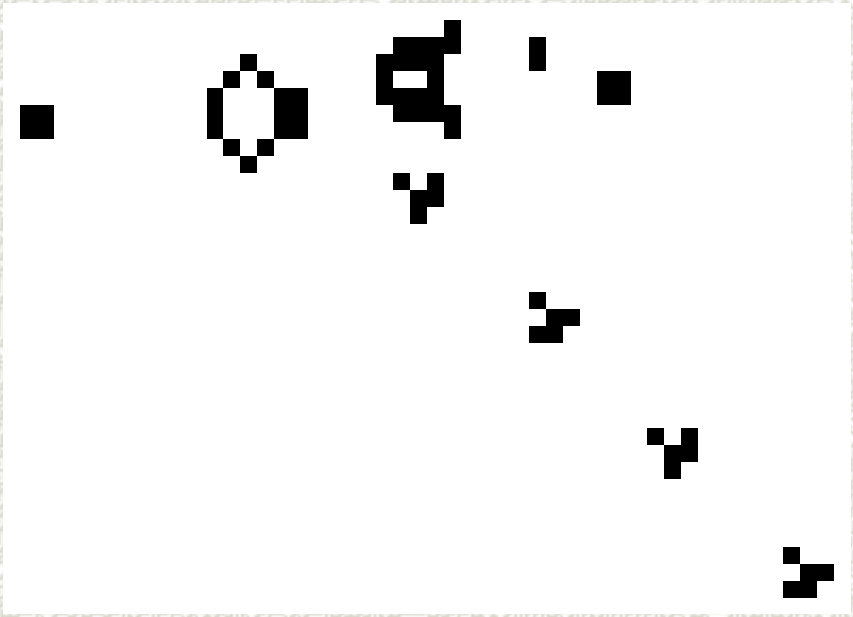
Universal Computation

- This showed that there were limits to what can be “computed”.
- Those limits are now well-known. Such a machine is said to be “Turing Complete” or support “universal computation” because they can compute everything that is computable or is capable of implementing any algorithm.



Cellular Automata

- Cellular Automata are a type of program in which *cells* are laid out on a grid. Each cell can be described in a finite number of *states*. Such states may be on/off. Green, Yellow, Red. The term Cellular Automaton is quite general.
- The states of cells are updated in an iterative process according to a rule.
- The states of cells are thus discrete in time.

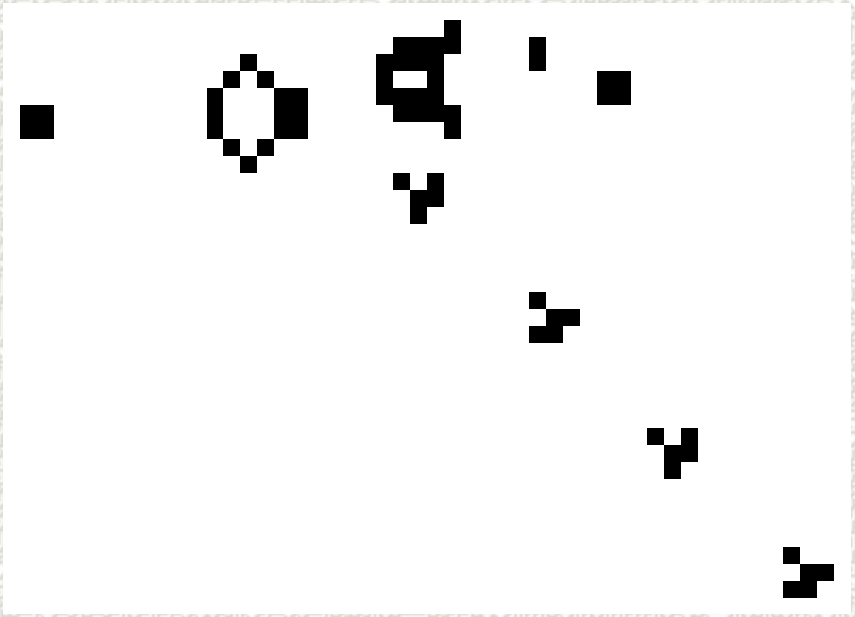


Cellular Automata

- Cellular Automata were created by John von Neumann in the 1940s. (ironically they are an example of non-von-Neumann style architectures) He was attempting to create a program that would have some of features of life. He created a cellular automata that would create a perfect reproduction of its initial state.

Cellular Automata

- One very famous example of a Cellular Automata (CA) is called Conway's Game of Life
- Conway Discusses Conway's Game of Life

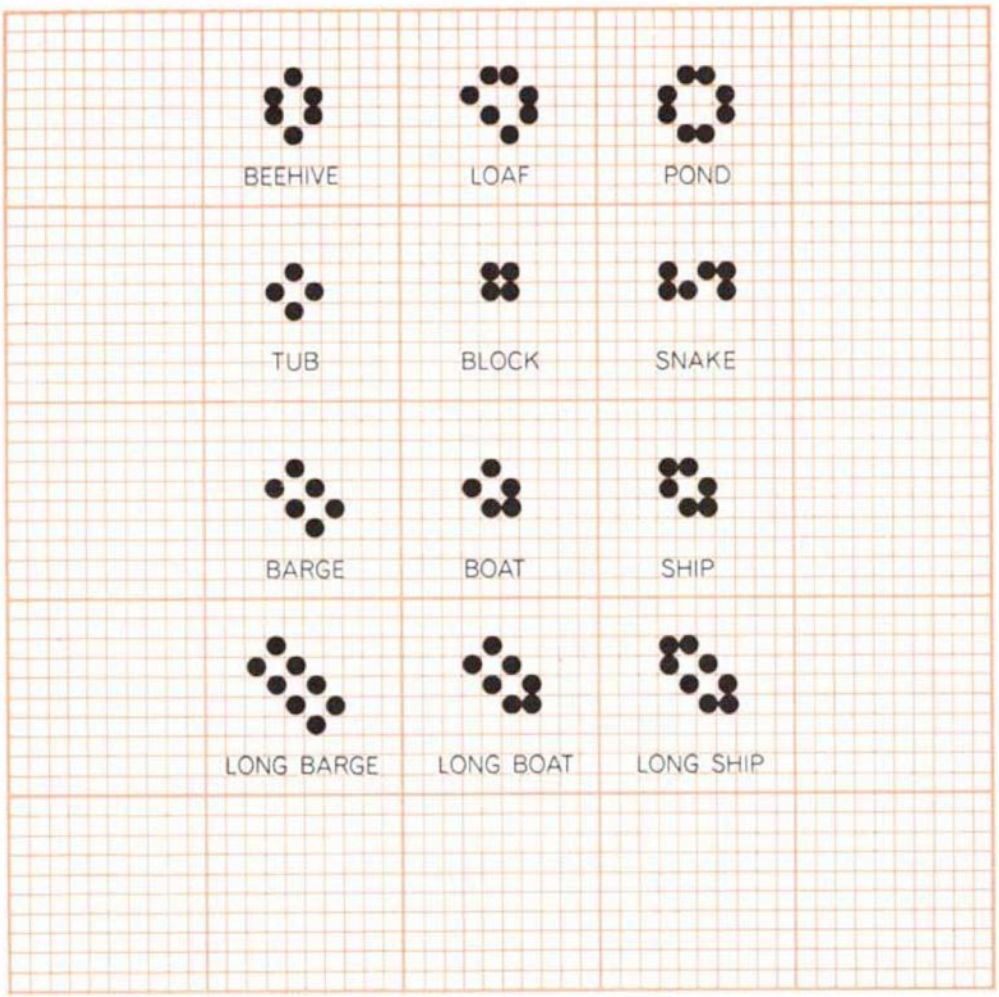


Game of Life Rules

- If a cell is ON
  - 0-1 ON Neighbors -> Off
  - 2-3 ON Neighbors -> On
  - 4-8 ON Neighbors -> Off
- If a cell is OFF
  - 0-2 On Neighbors -> Off
  - 3 ON Neighbors -> On
  - 4-8 ON Neighbors -> Off

Still Life

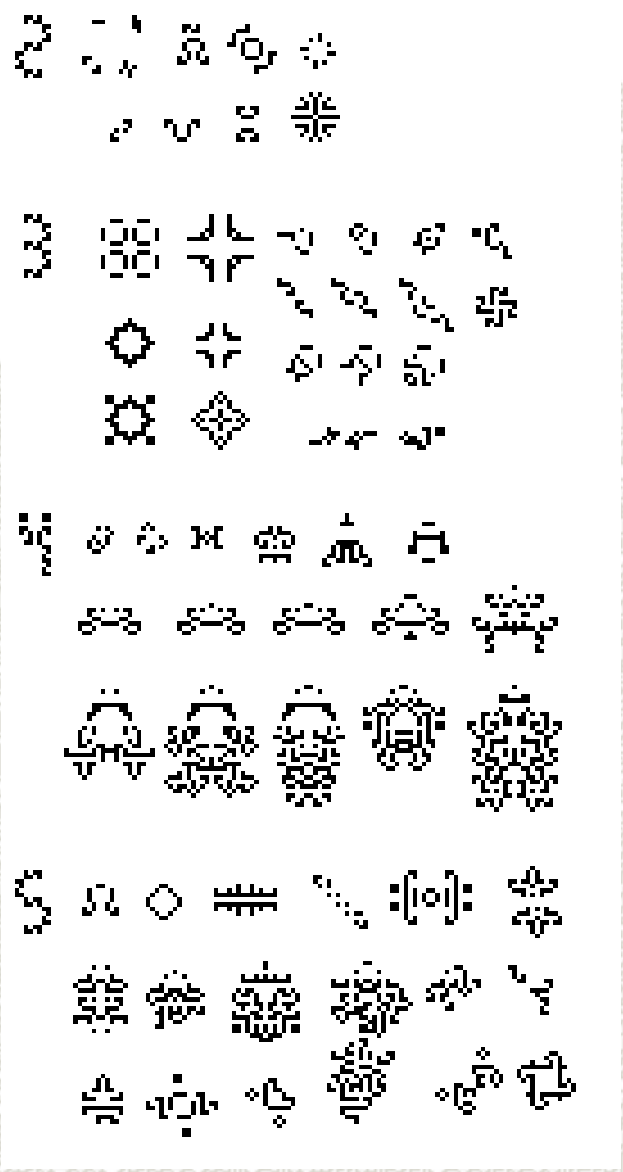
Still Life's are configurations in the Game of Life which are fixed in time. If we apply the rules to these configurations, all living cells remain alive and all dead cells remain dead.



The commonest stable forms

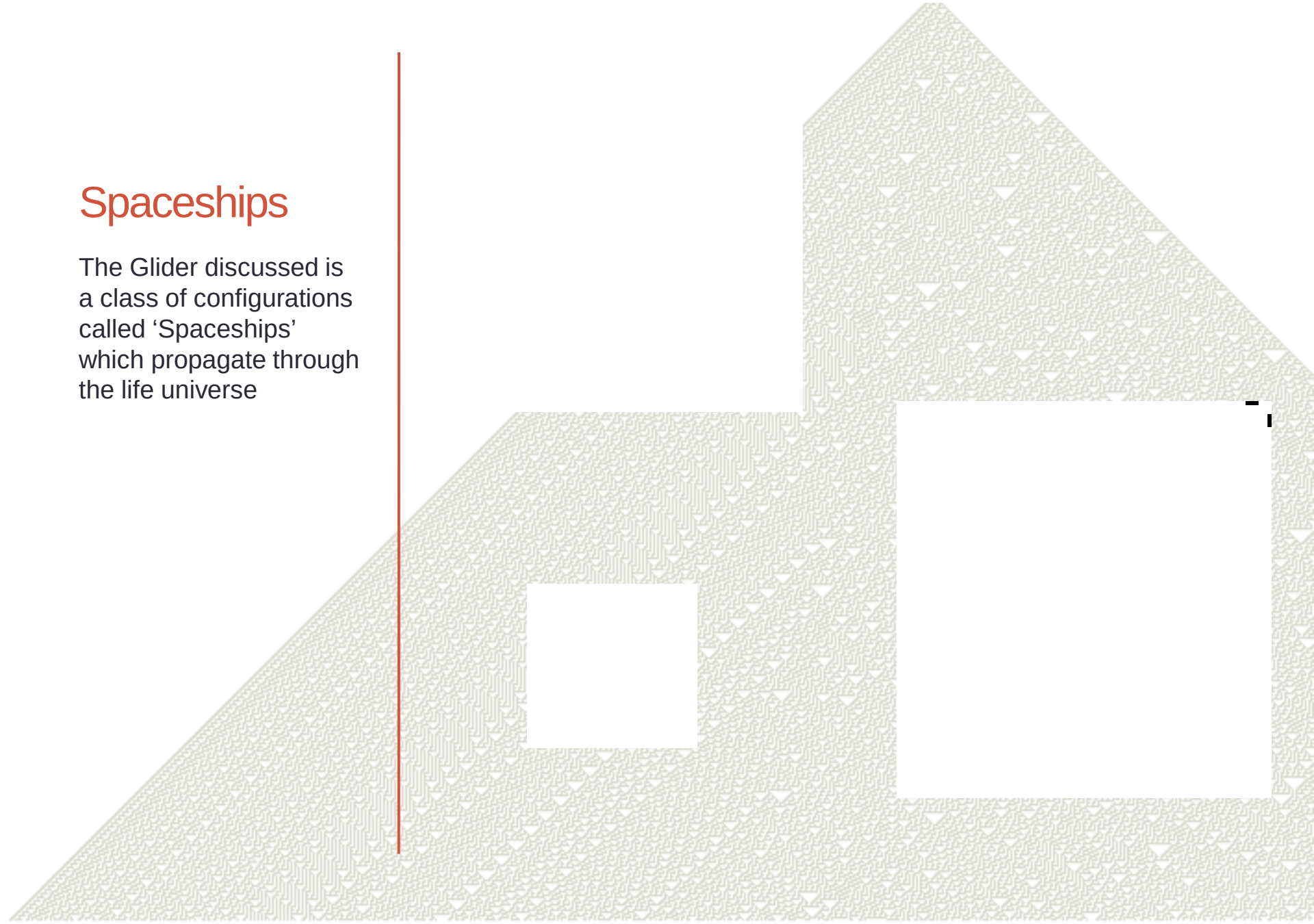
Oscillators

This configurations repeat after a set number of generations (iterations).



Spaceships

The Glider discussed is a class of configurations called ‘Spaceships’ which propagate through the life universe

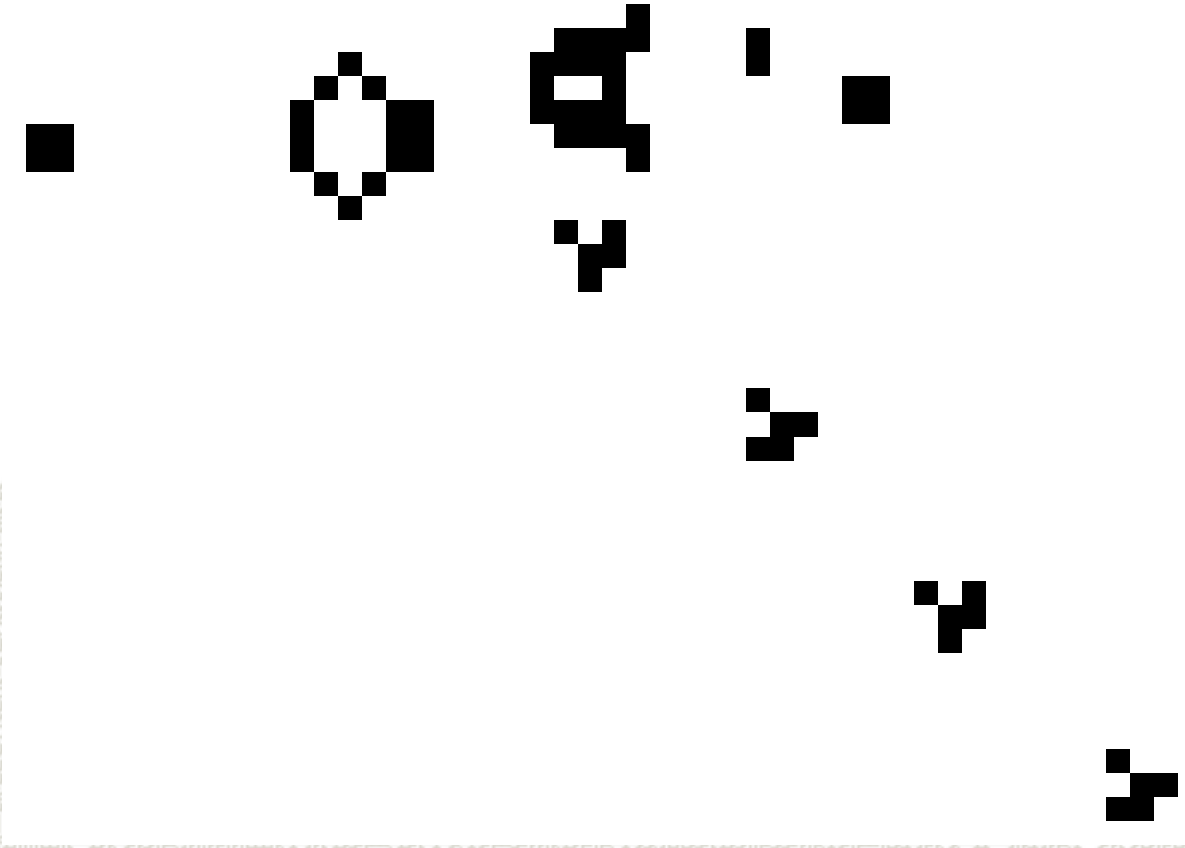


Guns

Guns are configurations that eject spaceships.

The Gosper's Glider Gun shown here was in response to Martin Gardner's famous column in *Scientific American* introducting Conway's Game of Life to a general audience

Bill Gosper received US$50 from Conway as a prize for discovering this.

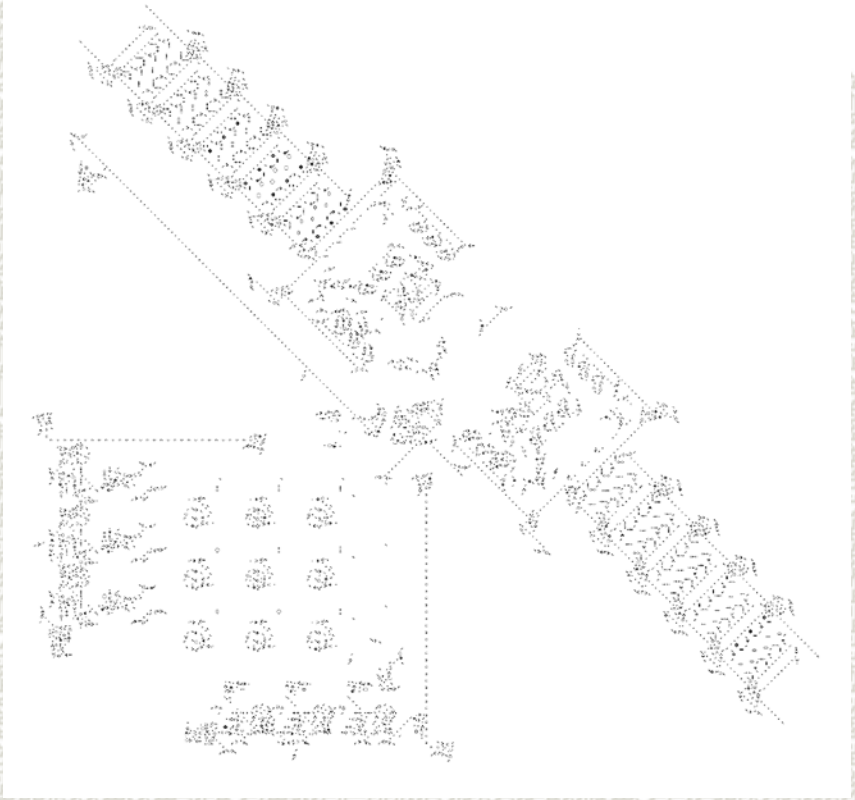


Game of LIfe

You may try drawing your own name in English or in hanzi

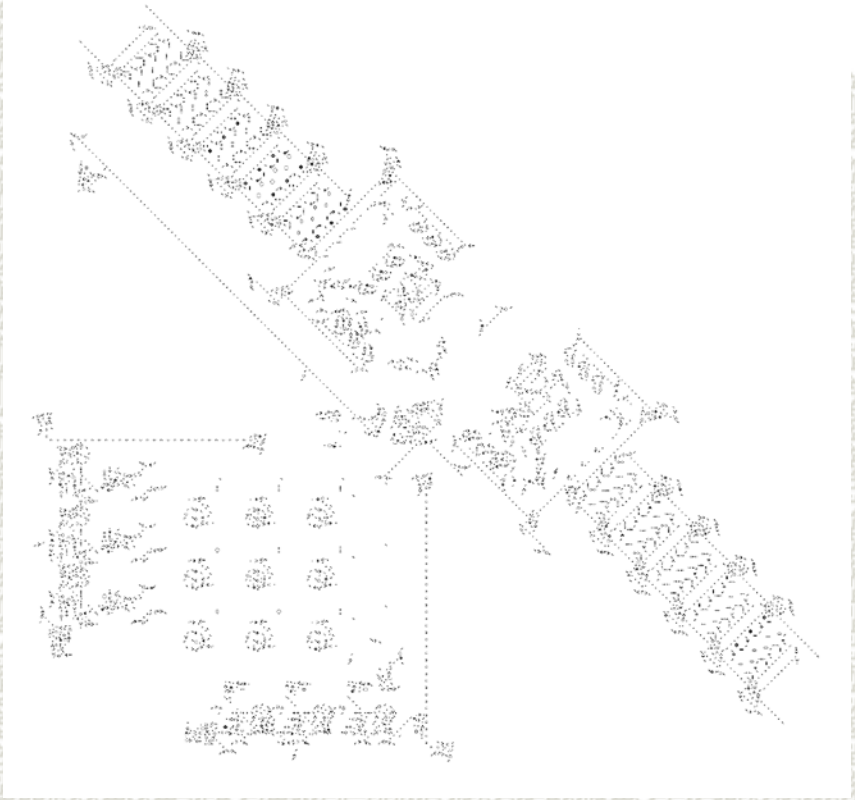
Game of Life

- Game of Life is capable of supporting Universal Computation
- A proof of this is beyond the scope of this course.
- It was shown that you could get things like gliders to interact in such a way that they acted like logical operations (AND, OR, XOR, etc.)



Game of Life

- Only hobbyists are interested in using Game of Life as a computer
- Any non-trivial calculation will require implementing many of these logical operations which isn't easy

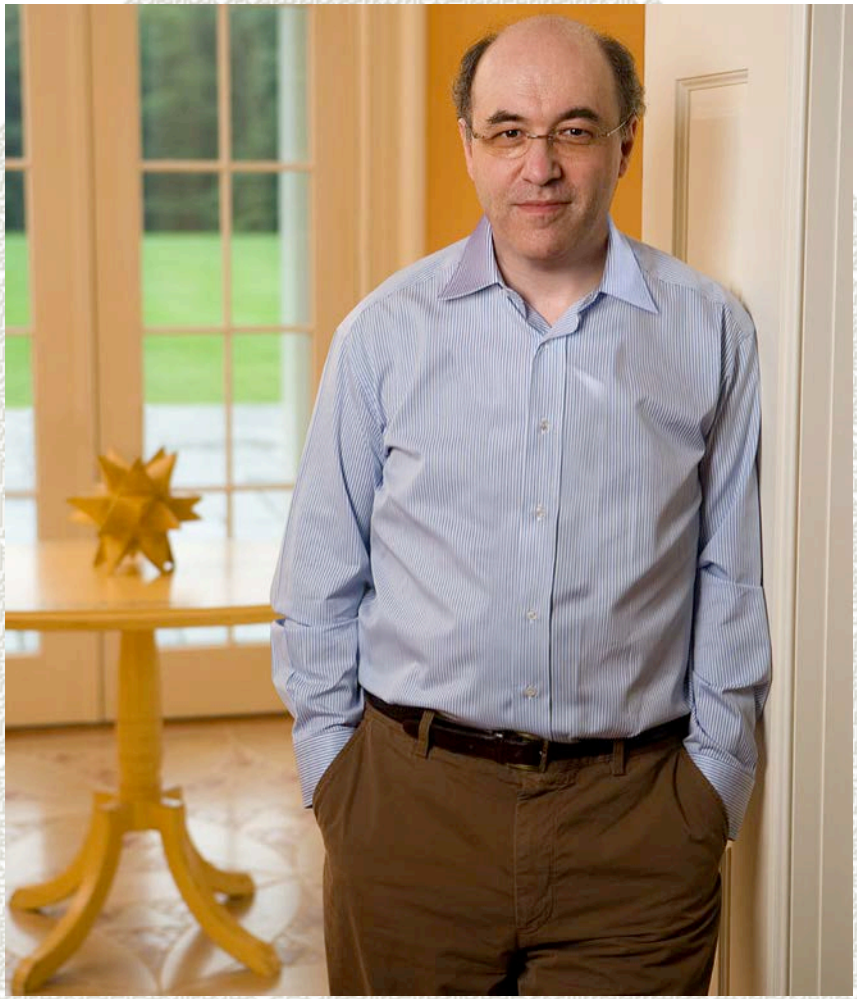


Game of Life

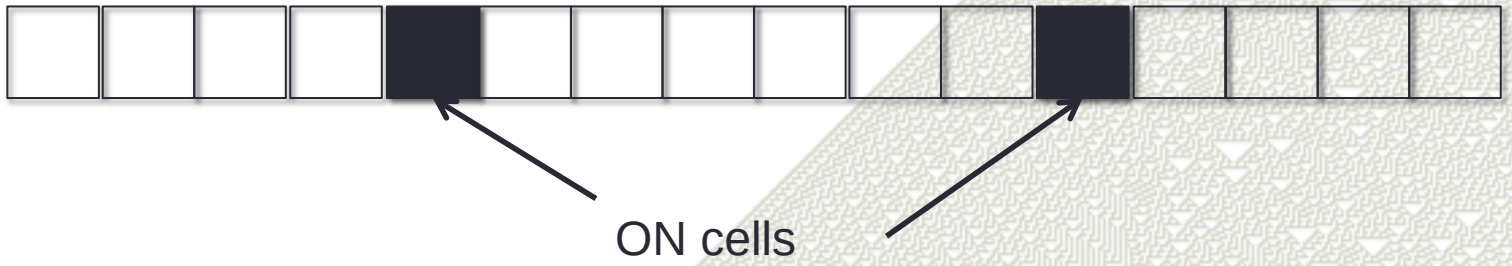
- Game of Life introduces a new style of computing.
- It is very hard to predict the outcome of random initial configurations.
- This is a statement about the possibilities.
- It is hard to predict because of how much it can do.

Elementary Cellular Automata

- Trying to understand how to use the power of this kind of computing Stephen Wolfram considered a simplified version of the problem he named *Elementary Cellular Automata*.

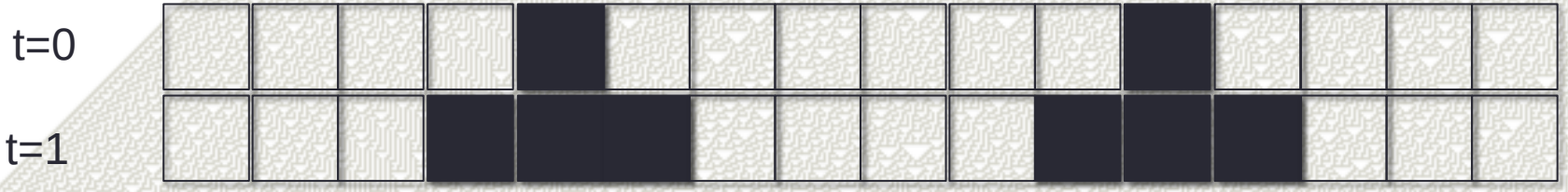


Elementary Cellular Automata

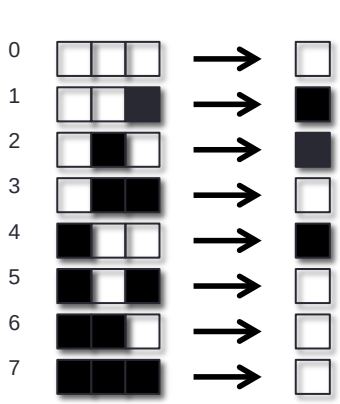


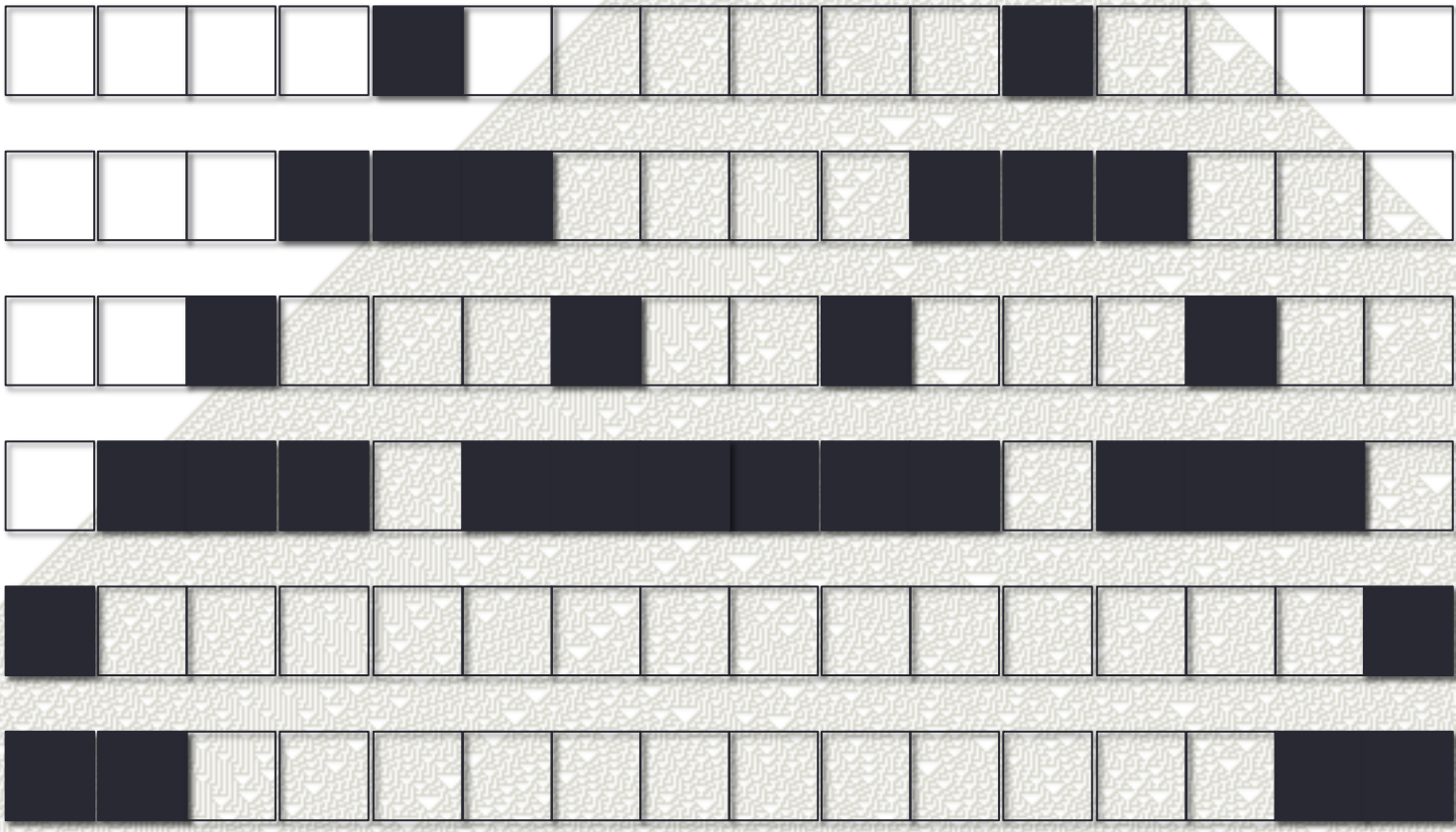
We look at a grid of cells. We can say that black is on and white is off.

Each cell will update itself in the next time step according to a rule.



Elementary Cellular Automata

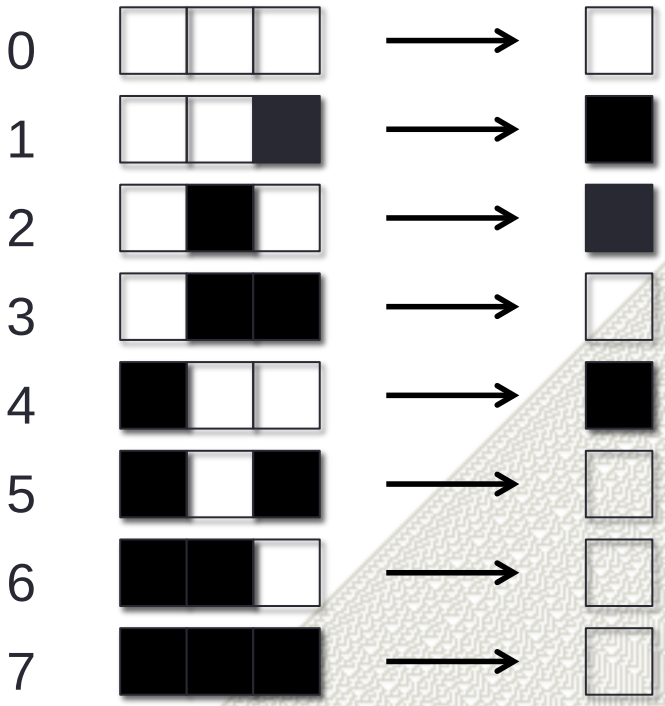




Numbering Cellular Automata

State	Binary Description	Base 10 Translation
	000	0
	001	1
	010	2
	011	3
	100	4
	101	5
	110	6
	111	7

Numbering Cellular Automata



We look at the update rule. For each number, x, on the left that updates to ON, we add 2^x to the Rule Number.

In this case, instructions 1, 2 and 4 update to ON.

This means the rule number is $2^1+2^2+2^4=2+4+16=22$

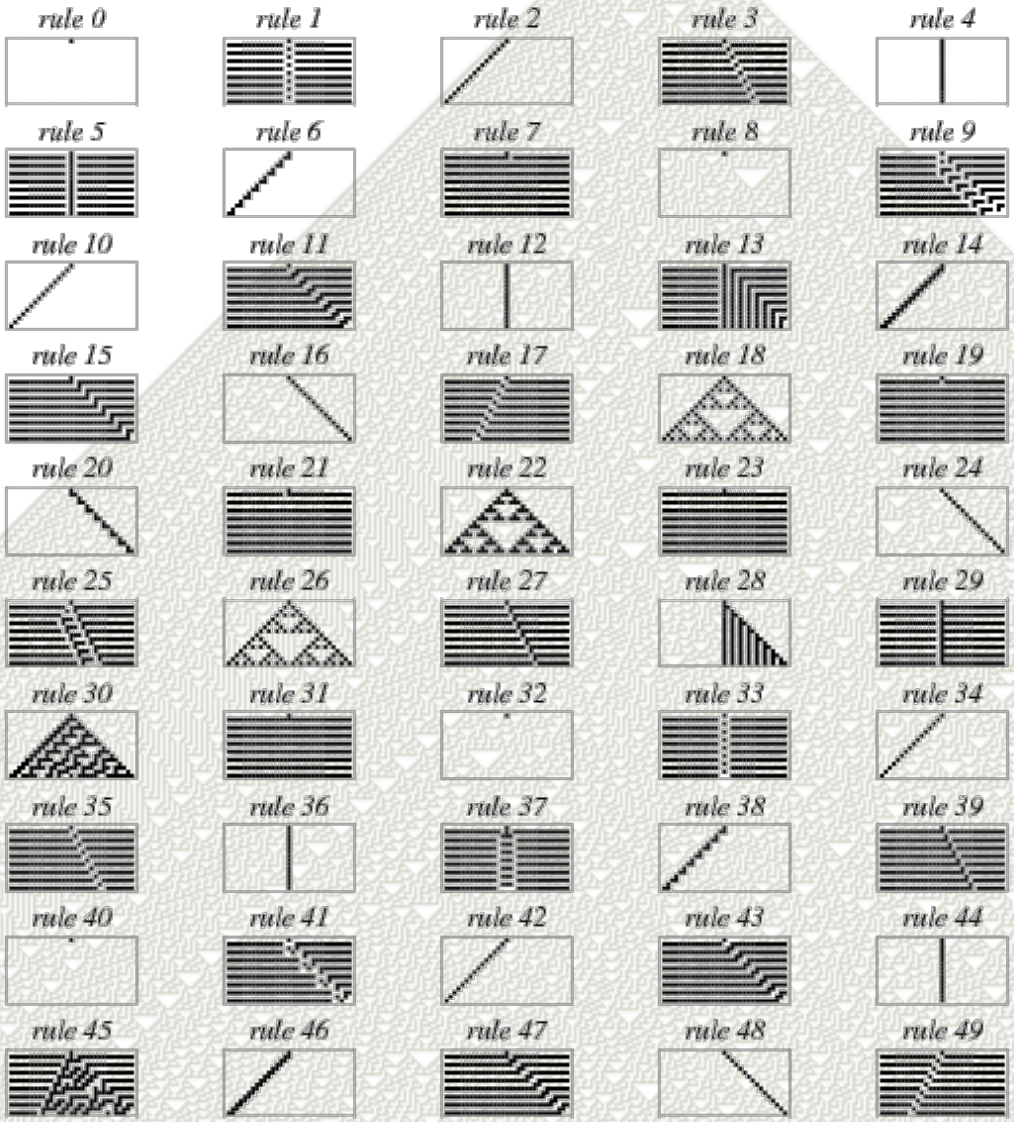
There are in principle $2^8=256$ Rules, Two choices for each of the 8 possible states of a cell and its nearest neighbors. In practice, some rules are equivalent up to the inversion of colors or of Left and Right.

Cellular Automata Rules

It is customary to show the rules from a "*single-bit initial condition*". This refers to a single ON cell in the center.

However, the behavior under a "*random initial condition*" may be more informative.

These are shown with a "*fixed boundary condition*" which assumes to the left and right of each image are imaginary cells which cannot be updated from the OFF state.

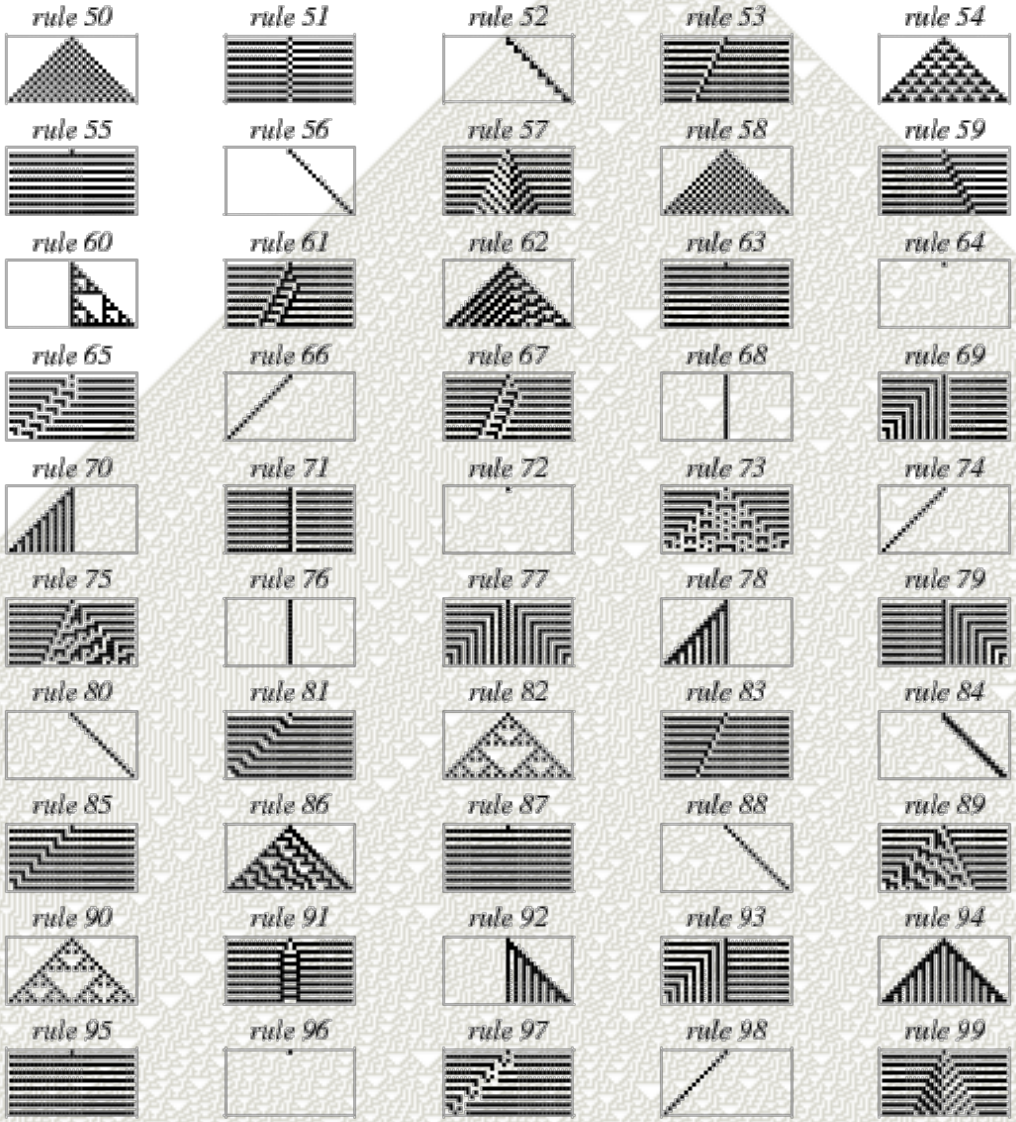


Cellular Automata Rules

It is customary to show the rules from a “single-bit initial condition. This refers to a single ON cell in the center.

However, the behavior under a “random initial condition” may be more informative.

These are shown with a “fixed boundary condition” which assumes to the left and right of each image are imaginary cells which cannot be updated from the OFF state.

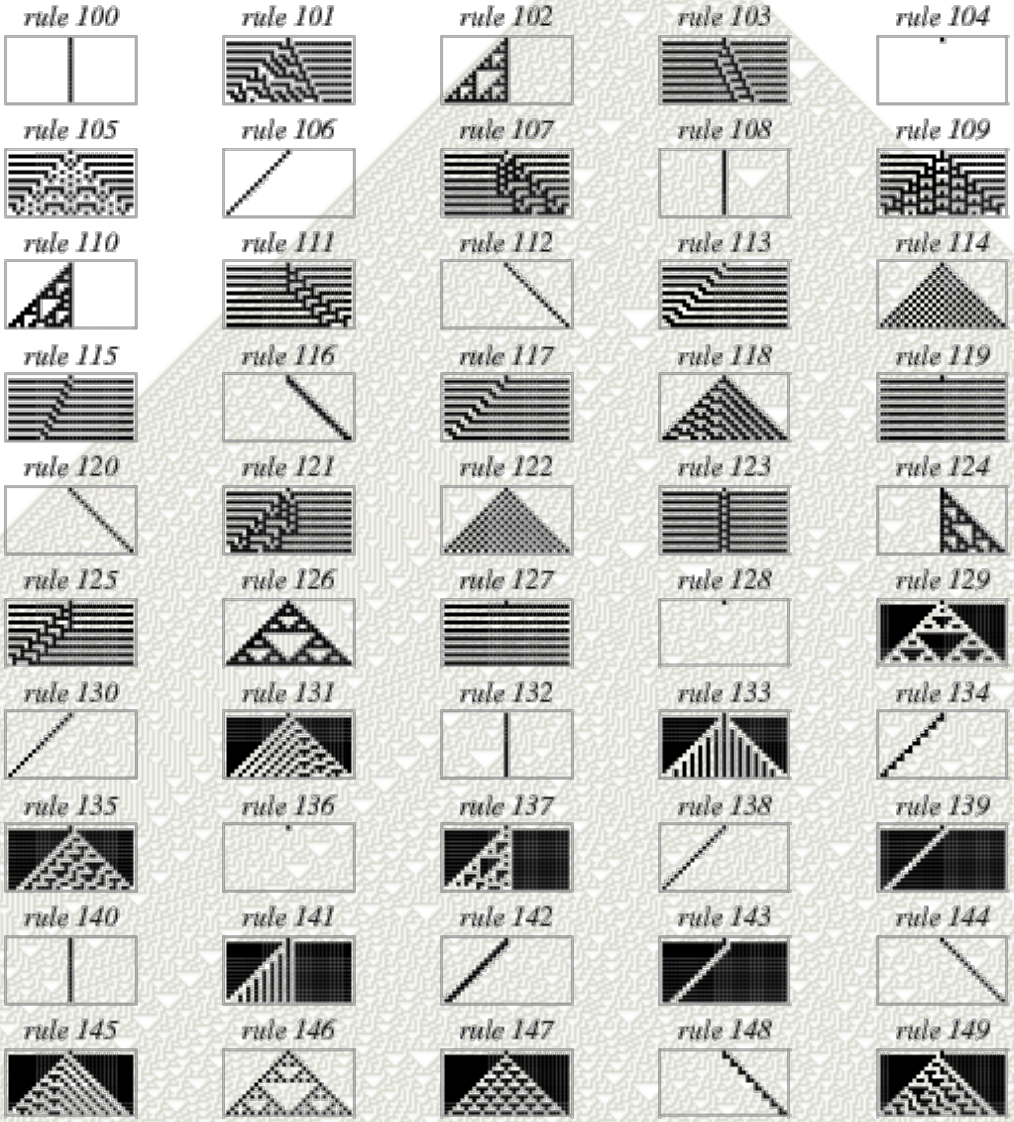


Cellular Automata Rules

It is customary to show the rules from a “single-bit initial condition. This refers to a single ON cell in the center.

However, the behavior under a “random initial condition” may be more informative.

These are shown with a “fixed boundary condition” which assumes to the left and right of each image are imaginary cells which cannot be updated from the OFF state.

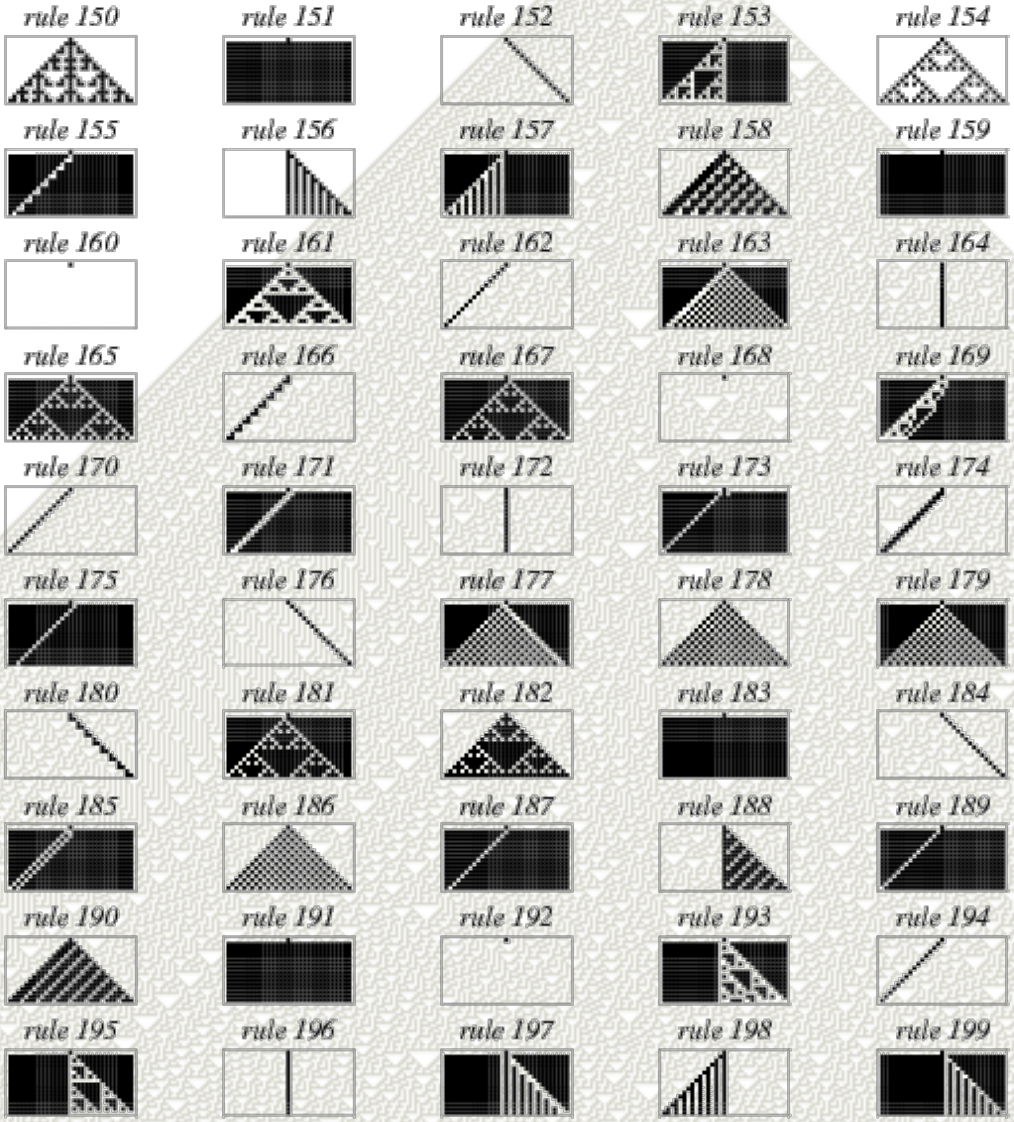


Cellular Automata Rules

It is customary to show the rules from a “single-bit initial condition. This refers to a single ON cell in the center.

However, the behavior under a “random initial condition” may be more informative.

These are shown with a “fixed boundary condition” which assumes to the left and right of each image are imaginary cells which cannot be updated from the OFF state.

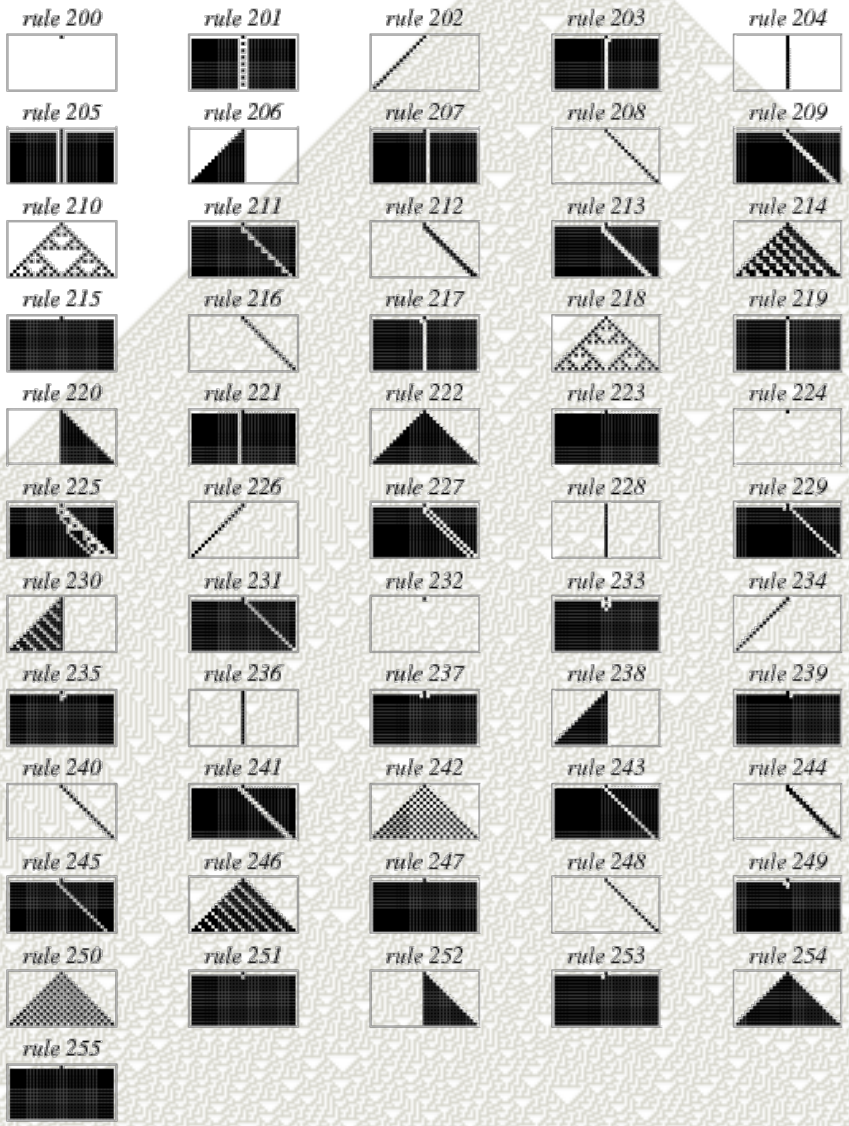


Cellular Automata Rules

It is customary to show the rules from a “single-bit initial condition. This refers to a single ON cell in the center.

However, the behavior under a “random initial condition” may be more informative.

These are shown with a “fixed boundary condition” which assumes to the left and right of each image are imaginary cells which cannot be updated from the OFF state.

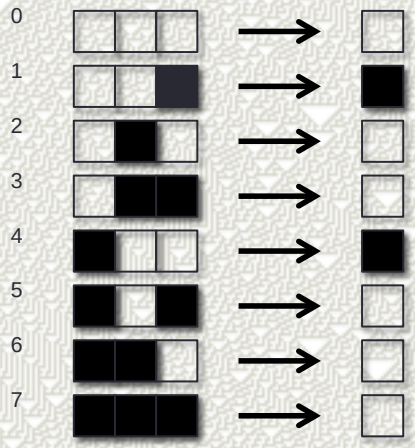


List of important jargon

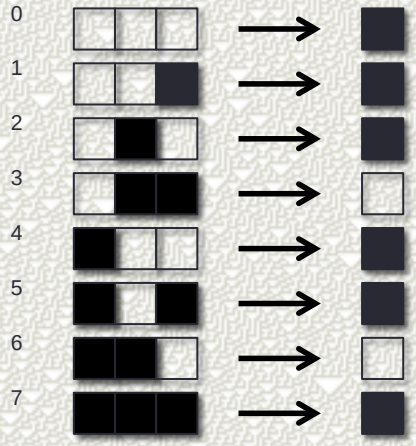
- Single Bit Initial Condition – the initial state of the CA is one ON cell and all the others OFF
- Random Initial Condition – the initial state of each cell in the CA is determined by chance.
- Fixed Boundary Condition – When we reach the edge of the CA, we assume that there is an invisible cell beyond the edge that is OFF (ON).
- Periodic Boundary Condition – There is an invisible cell beyond the left edge that has the same state as the right most cell. Similarly there is an invisible cell beyond the right edge that has the same state as the left most cell. (we can think of taping the two ends together in a ring so that there is no edge).

Classifying Cellular Automata

- Some rules are equivalent. They may differ only by an inversion of color or direction.
- There are only 88 unique rules.



Rule 18



Rule 183

This rule is equivalent to Rule 18 by switching to roles of the colors

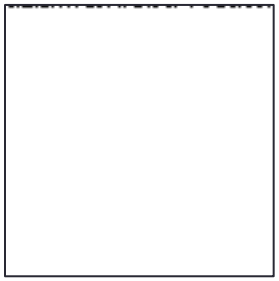
Classifying Cellular Automata

Class 1

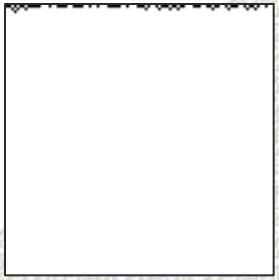
These Rules quickly converge to a homogeneous state from most initial conditions.

Very predictable

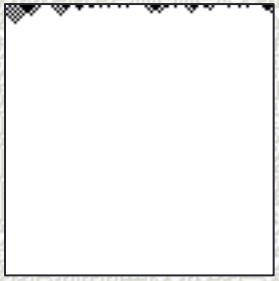
Very boring



Rule 0



Rule 32



Rule 160

Classifying Cellular Automata

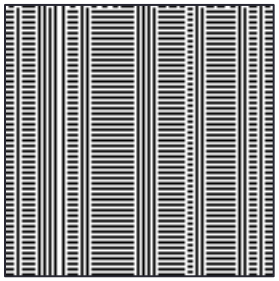
Class 2

These Rules quickly converge to a fixed or oscillating pattern.

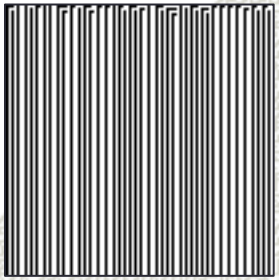
The details of the pattern itself may depend on the initial conditions.

Very predictable

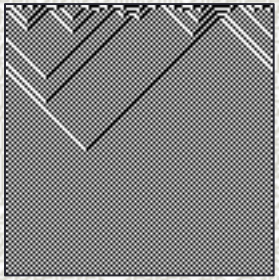
Only somewhat boring.



Rule 5



Rule 13



Rule 57

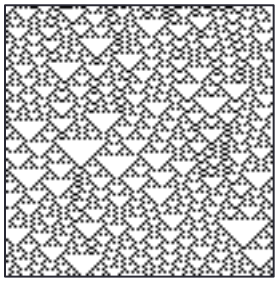
Classifying Cellular Automata

Class 3

These Rules display random patterns although some structures and patterns may emerge.

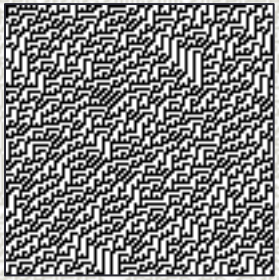
Not predictable

Not boring



Rule 18

Rule 30



Rule 57

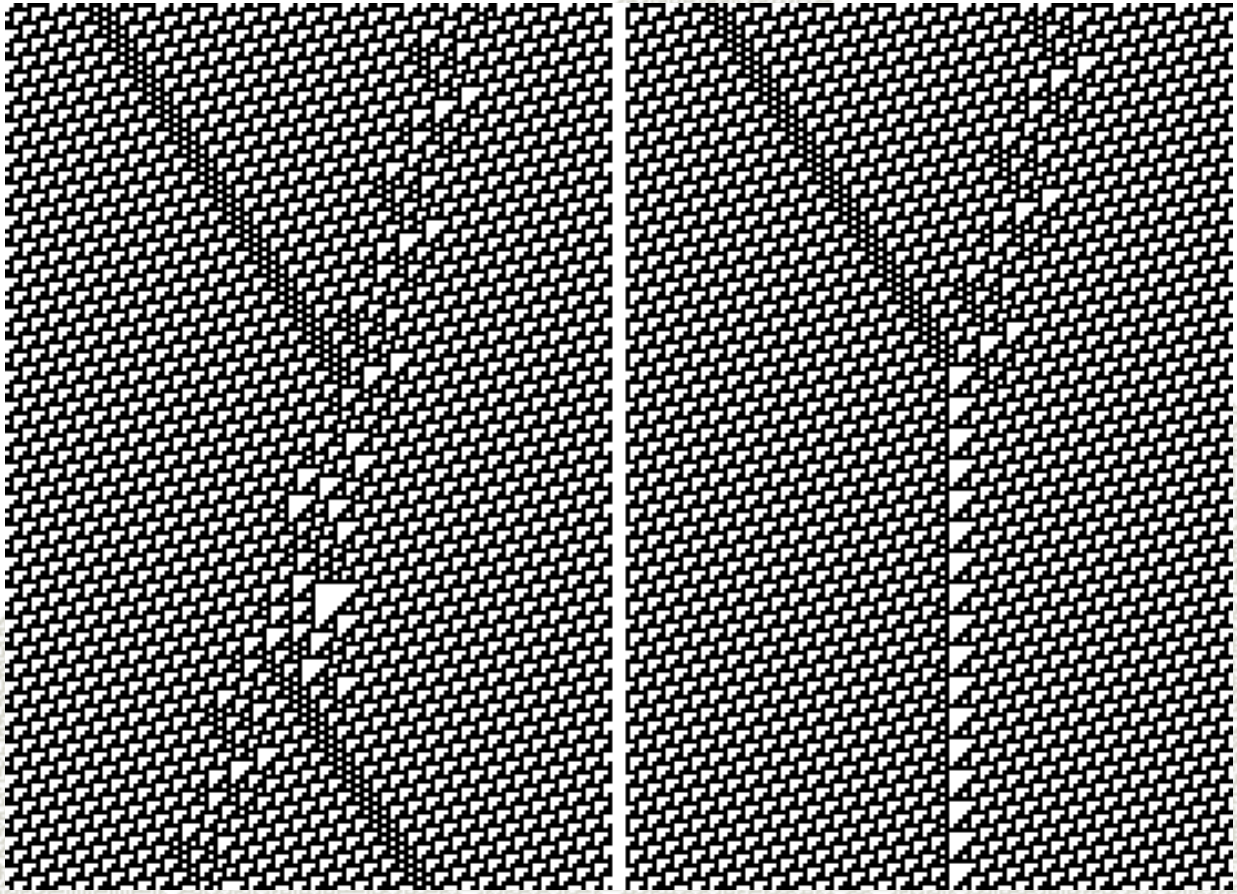
Classifying Cellular Automata

Class 4

These Rules display random patterns and emergent structures. Structures may collide and interact with each other in interesting ways.

Both predictable and NOT predictable

Most Interesting.



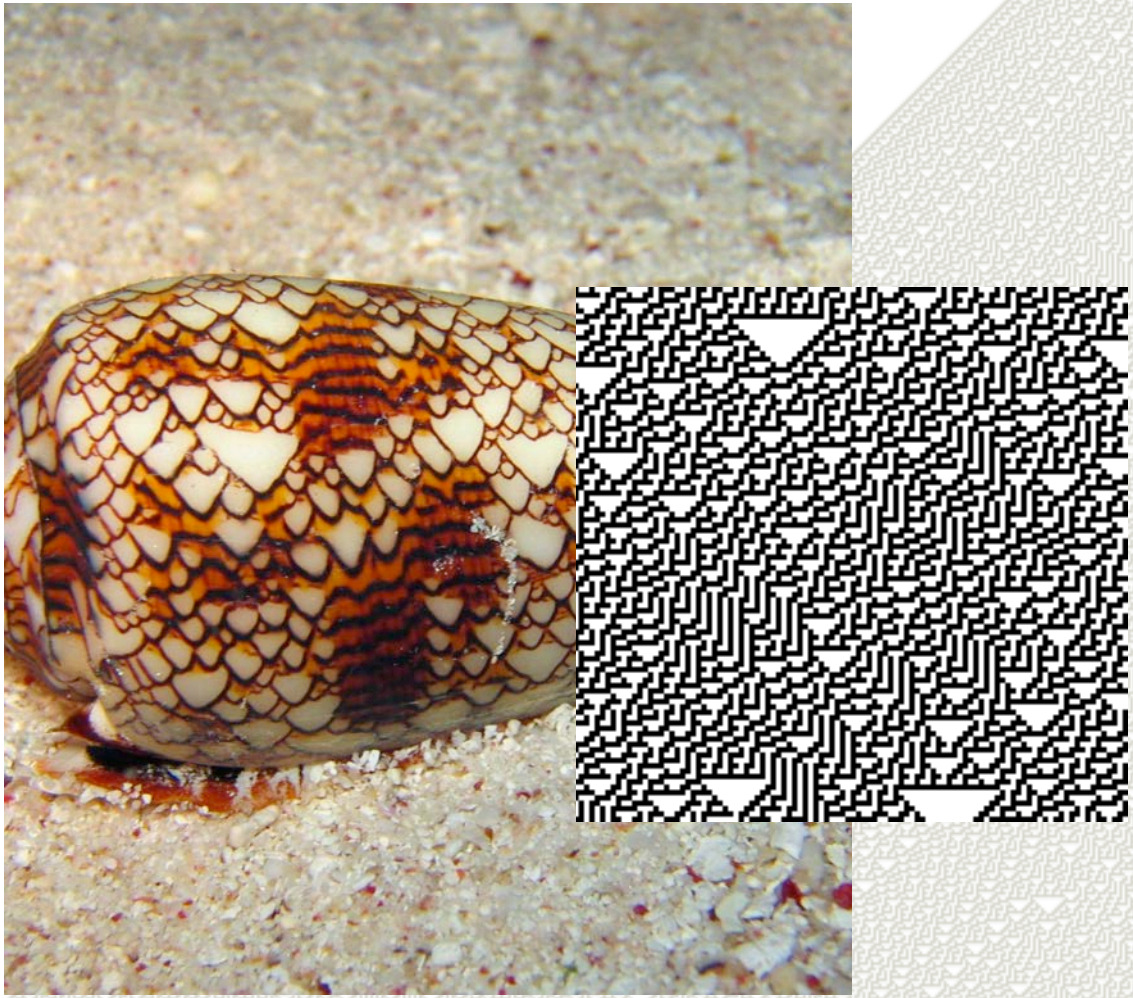
Rule 110

Why is Rule 30 so famous?

- Although there are other Class 3 elementary CAs with lower Rule numbers, Rule 30 shows a random looking pattern from the single bit initial condition.

- As highlighted in *Mitchell*, “The Rule 30 automaton is the most surprising thing I’ve ever seen in science…. It took me several year to absorb how important this was. But in the end, I realized that this one picture contains the clue to what’s perhaps the most long-standing mystery in all of science: where in the end the complexity of the natural world comes from.” -- Stephen Wolfram

Rule 30



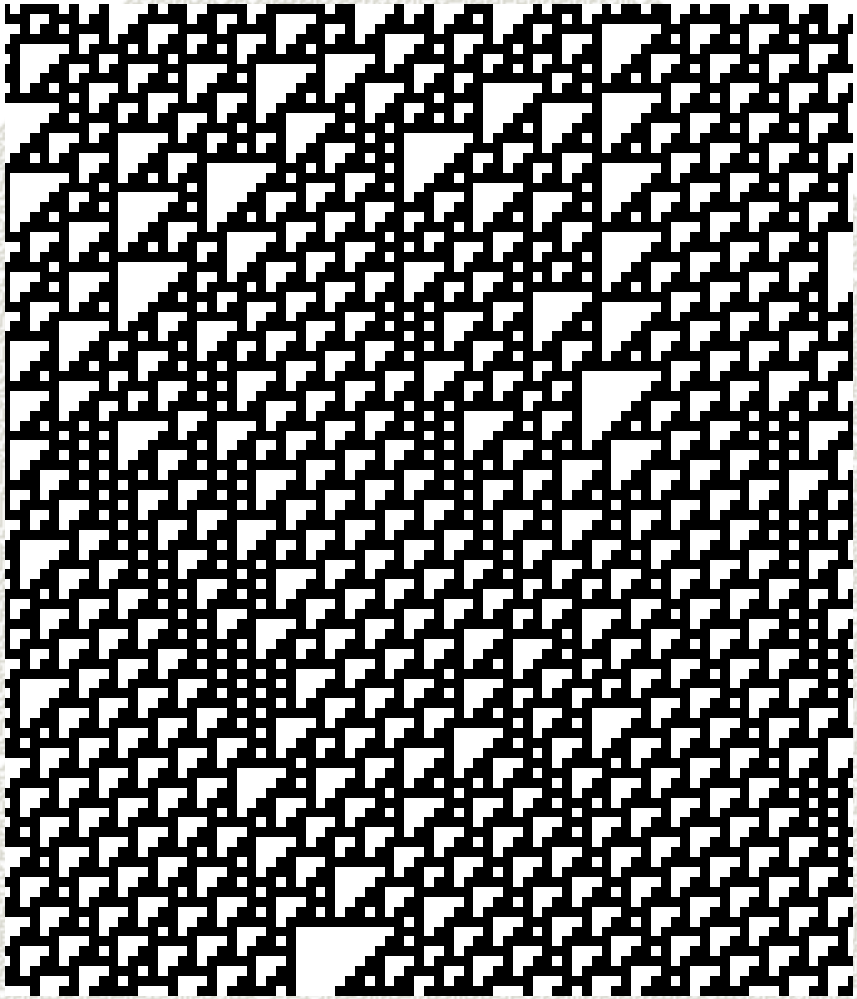
- The similarity in the pattern of this shell to the Rule 30 pattern indicates that the mechanism by which the shell colors itself need not be very complicated.

Rule 30

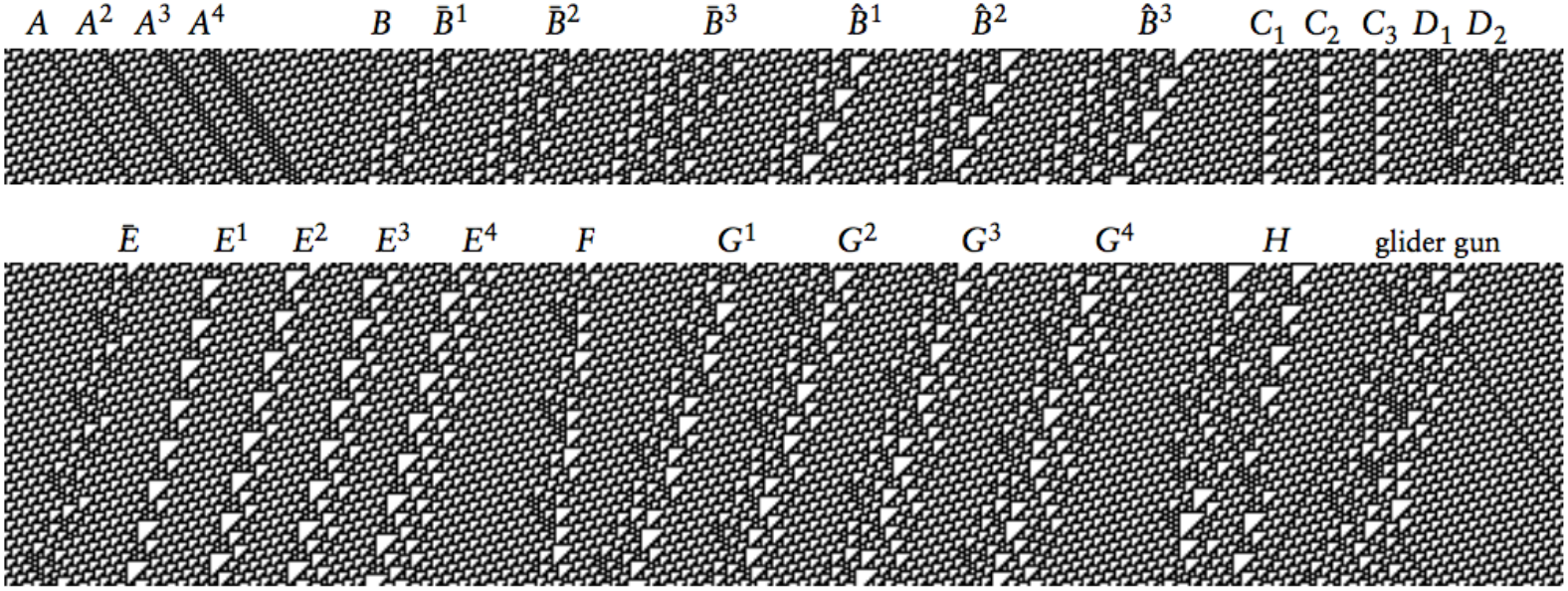
- Rule 30 creates a chaotic sequence in its center column
- Wolfram has patented this as a Pseudo-random number generator.
- If you ask WolframAlpha for a random number, it comes from Rule 30.

Class Four – Rule 110

- Wolfram speculated that there would be a relationship between Class Four CAs and support for “universal computation”
- This has been proved from Rule 110



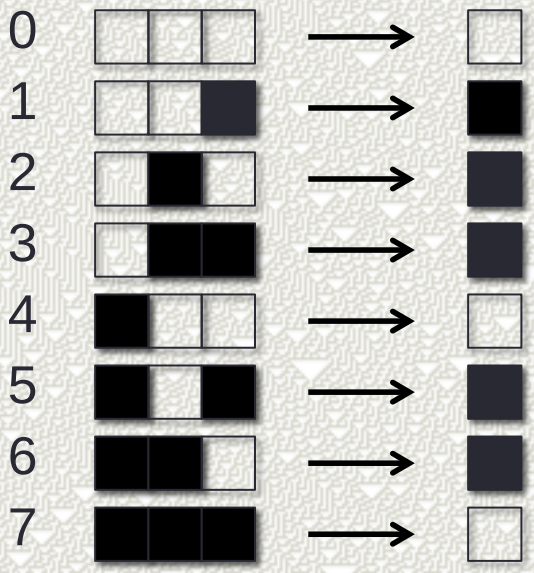
Rule 110 – Universal Computing



- Matt Cook proved that Rule 110 was capable of universal computation.
- Similar to the case with Conway's Game of Life, this was done by examining the interactions between "spaceship" or "glider" structures when they interact and making analogy to other logical operations.
- Above are some of the "spaceships" that can emerge in Rule 110

Simple Rules and Universality

- If a cell is ON
  - 0-1 ON Neighbors -> Off
  - 2-3 ON Neighbors -> On
  - 4-8 ON Neighbors -> Off
- If a cell is OFF
  - 0-2 On Neighbors -> Off
  - 3 ON Neighbors -> On
  - 4-8 ON Neighbors -> Off



Rule 110

Although we don't really know how to use them effectively, we have learned that Universal Computation may be born out of very simple rules.

Computational Reducibility

$$P_{n+1} = (1+c)P_n - D$$

0	1000000
1	980000
2	959900
3	939699.5
4	919397.9975
5	898994.9875
6	878489.9624
7	857882.4122
8	837171.8243
9	816357.6834
10	795439.4718
11	774416.6692
12	753288.7525
13	732055.1963
14	710715.4723
15	689269.0496
16	667715.3949
17	646053.9719
18	624284.2417
19	602405.6629
20	580417.6913
21	558319.7797
22	536111.3786
23	513791.9355
24	491360.8952
25	468817.6997
26	446161.7882
27	423392.5971
28	400509.5601
29	377512.1079
30	354399.6684
31	331171.6668
32	307827.5251
33	284366.6627
34	260788.496
35	237092.4385
36	213277.9007
37	189344.2902
38	165291.0117
39	141117.4667
40	116823.0541
41	92407.16932
42	67869.20517
43	43208.5512
44	18424.59395
45	-6483.283079

In Problem Set 2A, you are asked to calculate the size of a debt by iterating the function above two times.

However, there is a “closed” form solution.

$$P(k) = (1+c)^k\left(P_0 + \frac{D}{c}\right) - \frac{D}{c}$$

If I don’t have this “closed” form solution, knowing the value of the debt after 100 months, requires 100 calculations.

If I DO have this “closed” form solution, knowing the value of the debt after 100 months, requires a SINGLE calculation.

This closed form reduces the computational complexity of the problem.

Computational Irreducibility

- A closed form equation for the n^{th} step does not always exist.
- You may be able to take advantage of some patterns to reduce the computation somewhat.
  - For Gosper's Glider Gun, it's easy to know what the 110^{th} time step will look like because we know the periodicity of the gun….
  - Some similar simplifications can be made for Rule 110 or in Rule 30
  - These are not generally true.
- Class 3 and 4 Cellular Automata show aspects of Computational Irreducibility.
- The only way to know what the system will look like in the future is to let it run.

Computational Equivalence

1. Processes in Nature should be thought of as “computation”.
2. Since even very simple rules can obtain universal computation, the ability to support universal computation should be common in nature.
3. Complexity in nature is limited by computability. Or no process in nature produces a non-computable behavior.
4. All processes which are not obviously simple are equivalent in complexity.

- Your brain, your smartphone, the weather, Biological Evolution, Rule 110…all working calculations of equivalent complexity!

Computational Equivalence

1. Processes in Nature should be thought of as “computation”.

- At any given point in time, a cell in an automaton processes *information* about its state and the state of its neighbors.
- The rule need not be complicated in order for the cell to perform an interesting task.
- This is the key to *self-organization*.
- Models of natural systems based on this general idea have been shown to demonstrate emergence.

Computational Equivalence

2. Since even very simple rules can obtain universal computation, the ability to support universal computation should be common in nature.

- Although it has been shown that many models of natural systems can be thought of as information processing or computing, it really is an open question whether or not these can support universal computation, (win Nobel Prize here).
- However, it seems given I can fit two universal computation rules on a single powerpoint slide that this is not implausible.

Computational Equivalence

3. Complexity in nature is limited by computability. Or no process in nature produces a non-computable behavior.

- For reasons beyond the scope of this course, this is true if reality is digital. If the universe is fundamentally coarse-grained, then Turing's proof applies to all machines. (win Nobel Prize here)
- Again this is an open question

Computational Equivalence

4. All processes which are not obviously simple are equivalent in complexity.

- This one is really tricky to know. Is my brain doing calculations that are equivalent in complexity to the weather?!
- However, if that is true it might lead us to believe that the Universe is governed by a simple Cellular Automata-like rule.
- “I don’t know. In *Mathematica*, for example, perhaps three, four lines of code.” – Stephen Wolfram
- Stephen Wolfram TED talk

Other philosophical thoughts…

- If you were a scientist living in a Game of Life-like universe, would you come up with the rules of your nature?
- If you did would it necessarily be the same as Conway's rules?
- What if the initial state was a Glider Gun, Still Life, Oscillator…?

Other philosophical thoughts…

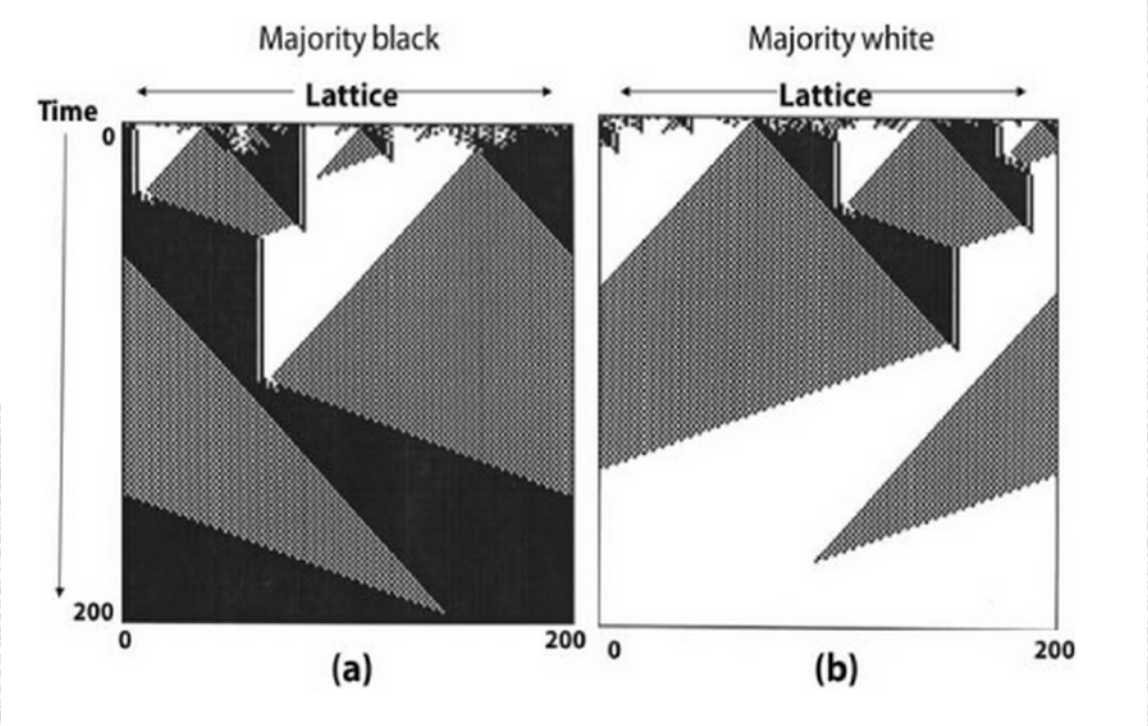
- The textbook author Mitchell and a collaborator tried to solve something called the majority classification problem.
- The idea is to come up with a CA that can tell whether it's initial state is mostly ON or mostly OFF.

Their first guess at the problem came up with something like this….

(They were using slightly more complicated CA than we have discussed)

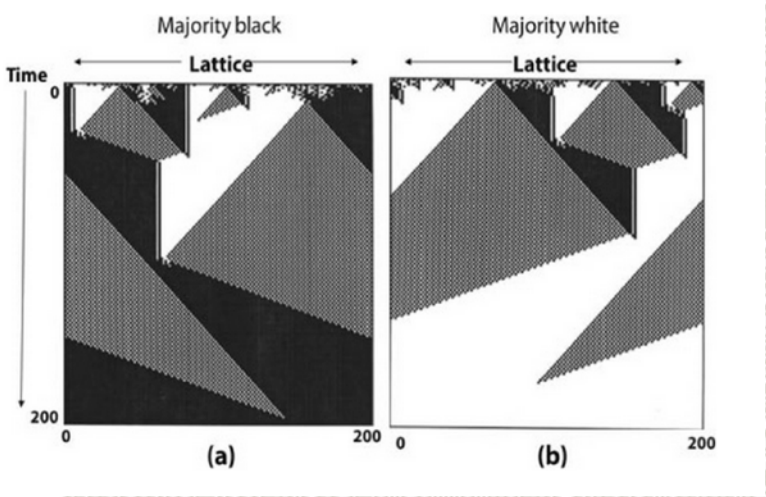
Other philosophical thoughts.

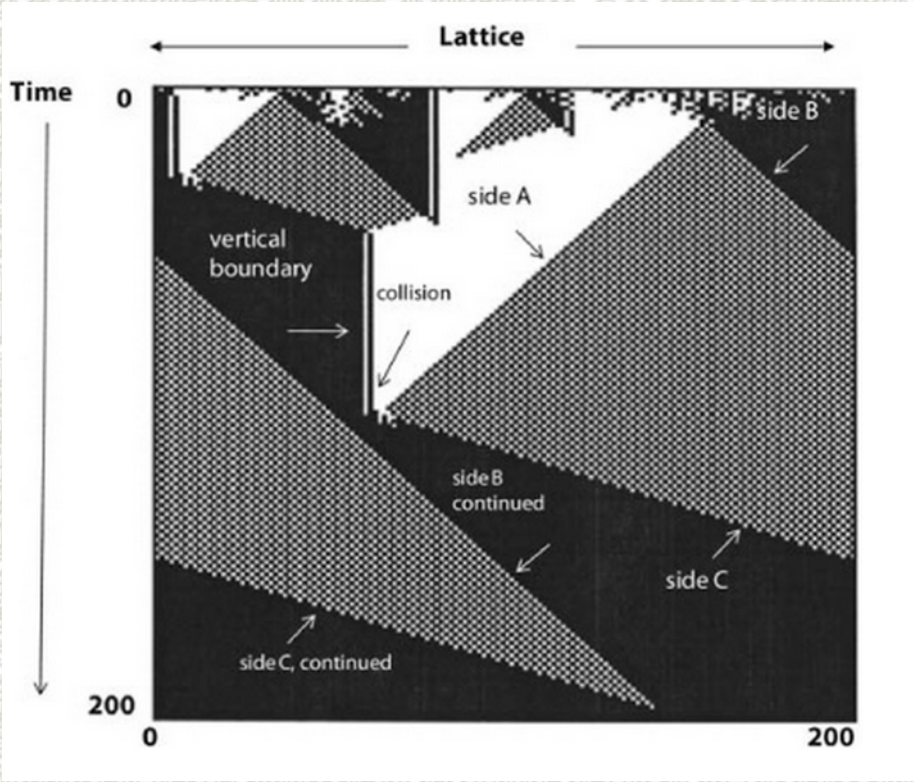
- They began exploring the space of all possible rules to find the one that could best do the job untill they came up with this…



Other philosophical thoughts.

- The solution was not intuitive to them but they could sort of reverse engineer the concept. Crutchfield thought that they represented information processing structures.





Other Philosophical Thoughts…

- He gave the process as sort of “particle” description
- Both work equally well. We may prefer the description on the right… However this is a description imposed by the scientists. Our current theories of QM, GR may be similar descriptions of an underlying CA-like computation