The World of Simple Programs

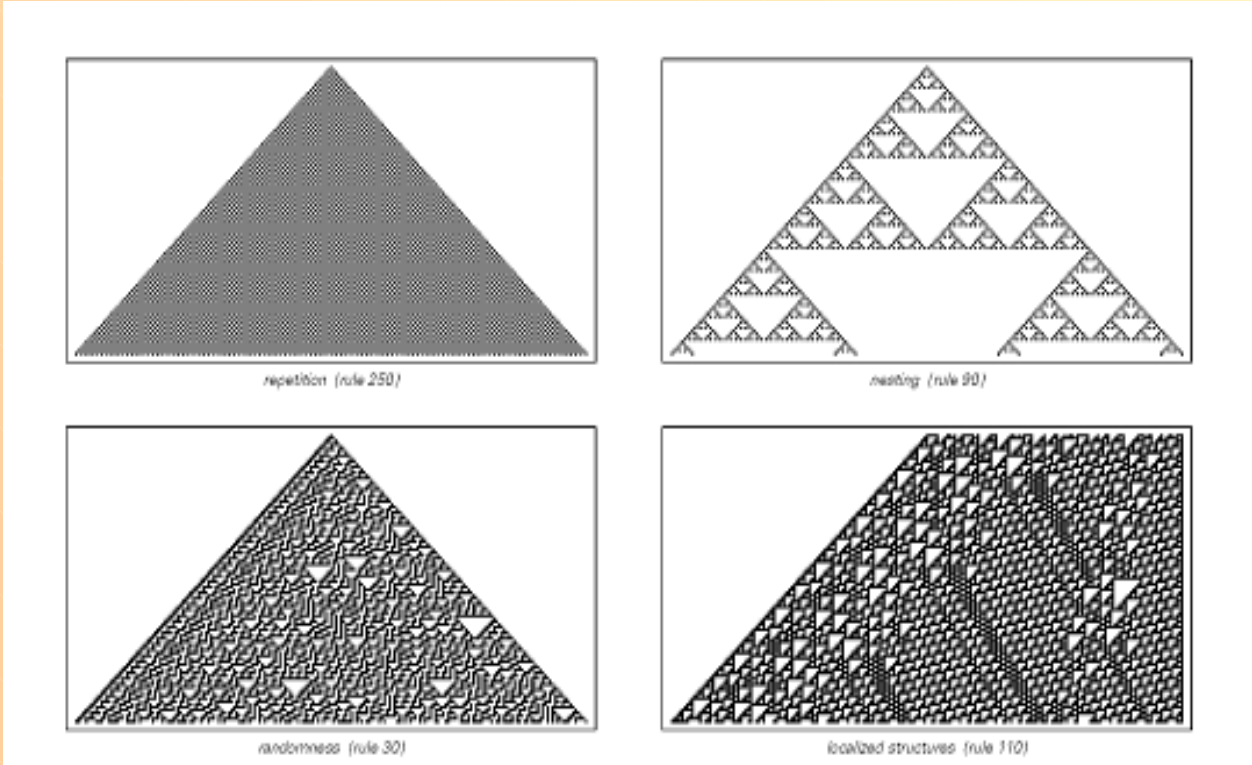
From *A New Kind of Science*
by Stephen Wolfram

Presented by Adam Olenderski

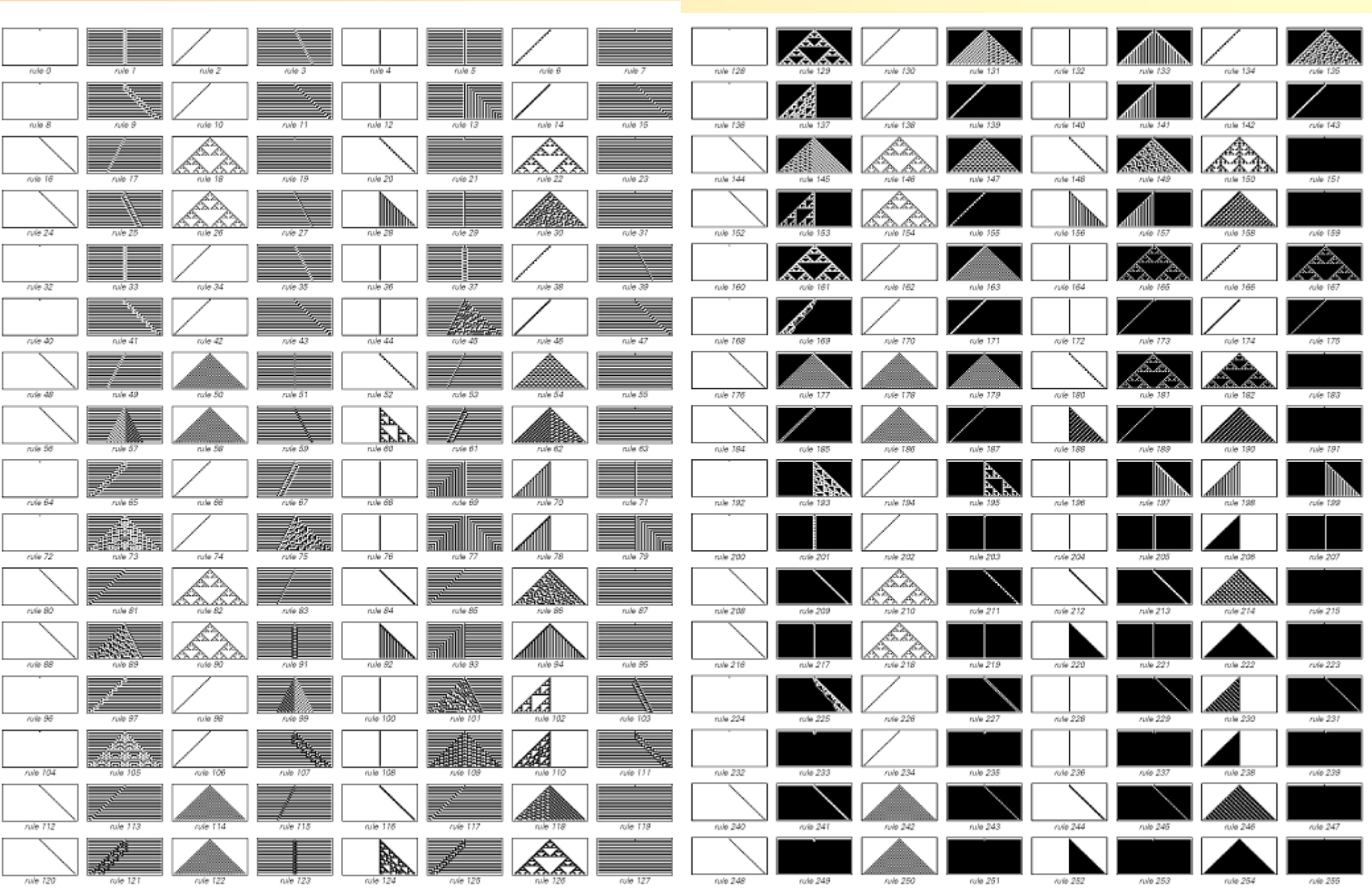
CS 790R, University of Nevada, Reno
1/30/2006

The Search for General Features

- Is it only cellular automata with very specific rules that produce apparent complexity?
- If not, what other kinds of programs can produce similar results?

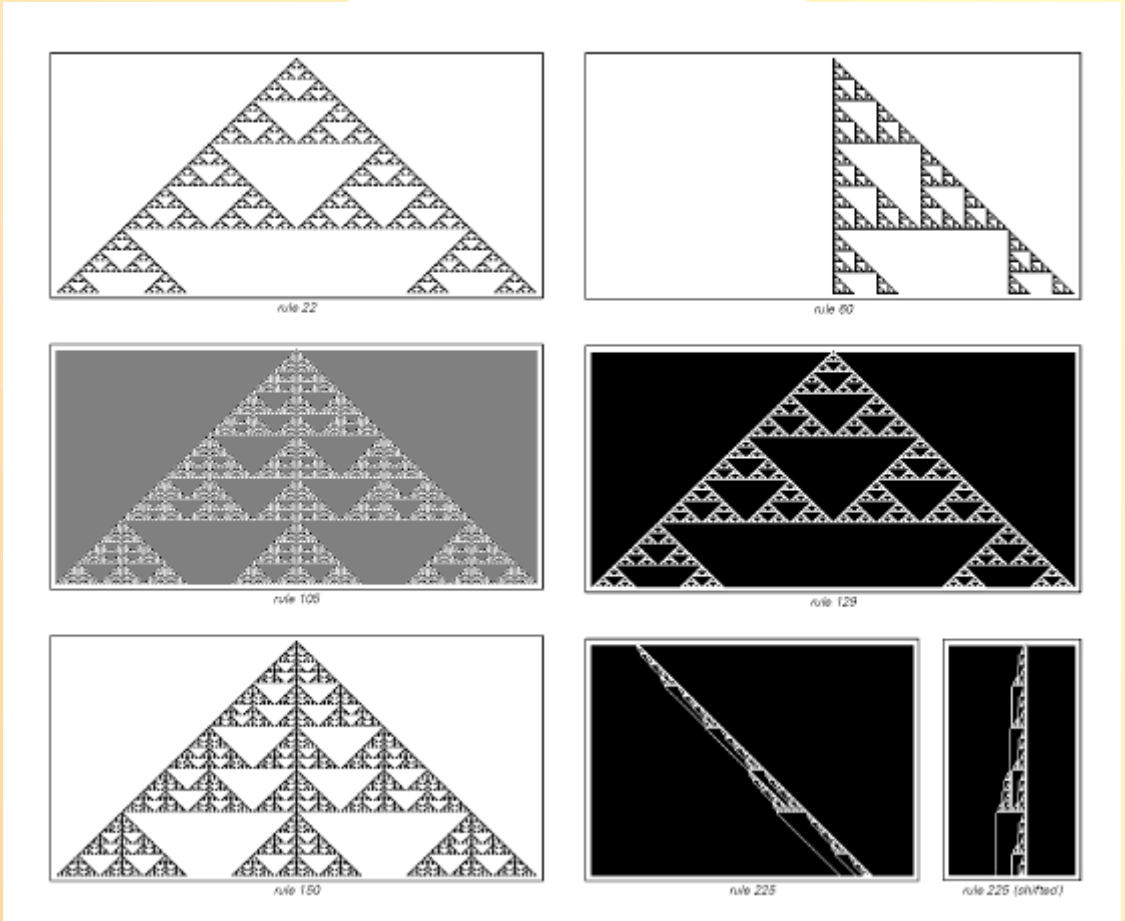


More Cellular Automata



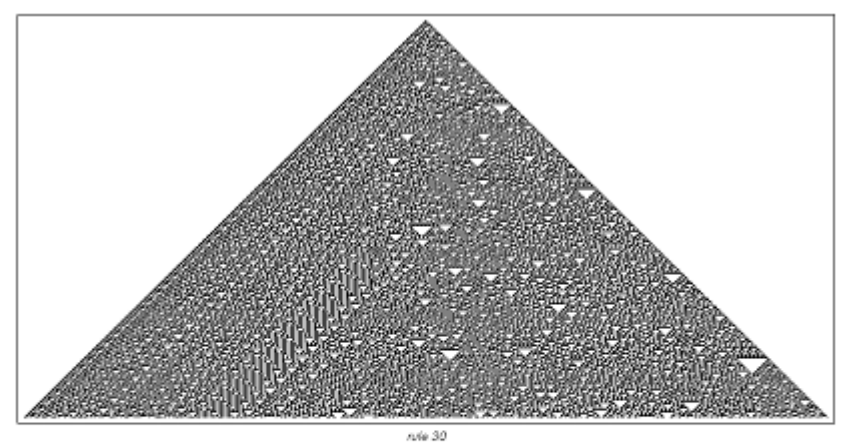
More Cellular Automata

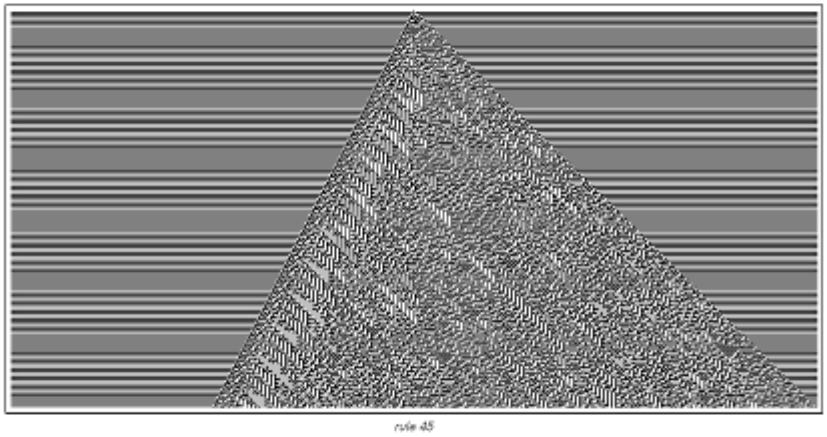
- Repetitive patterns are slightly more common
- About 14% are more complicated (nested or fractal structures, apparent randomness)
- Of these, most are nested patterns (e.g., the “Triforce” pattern and variations)

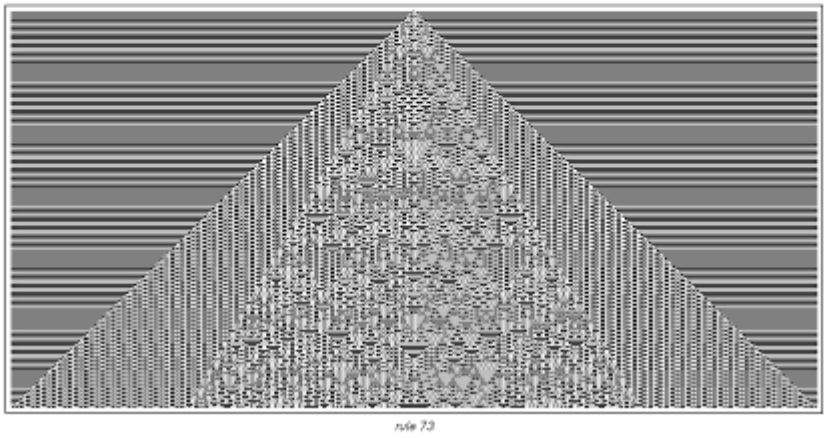


More Cellular Automata

- 10 of the 256 yield apparent randomness
- 3 basic forms

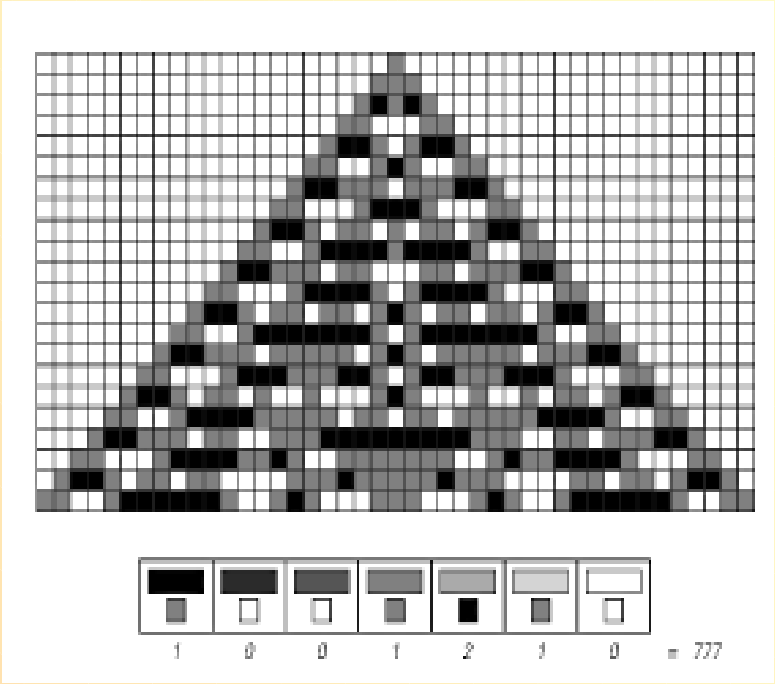






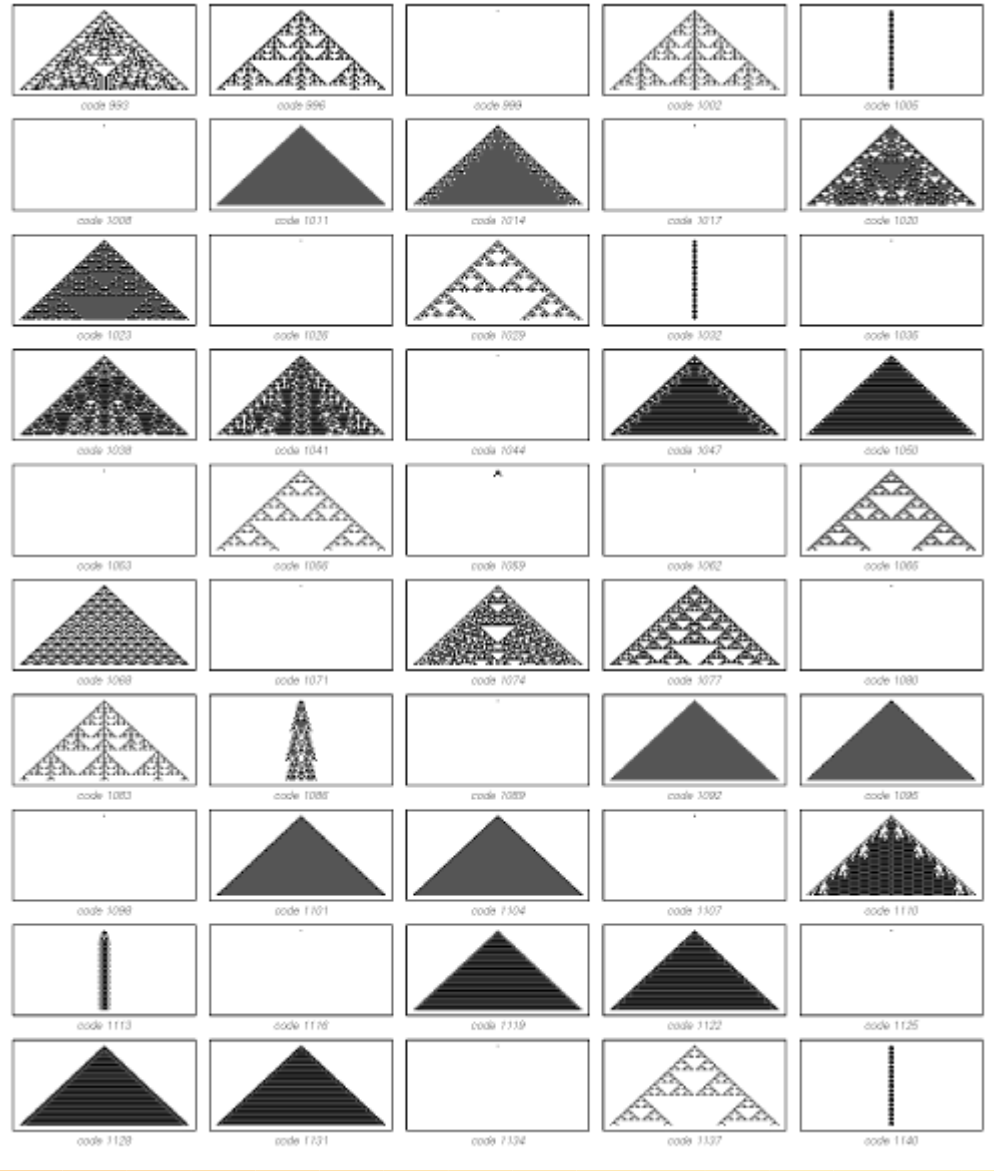
More Cellular Automata

- What if we increase the complexity of the rules by adding a third color?
- Number of possible rules increases from 256 to 7,625,597,484,987.
- Reduce this to 2,187 by using totalistic rules, which take new cell color from the average color of neighboring cells instead of individual colors.



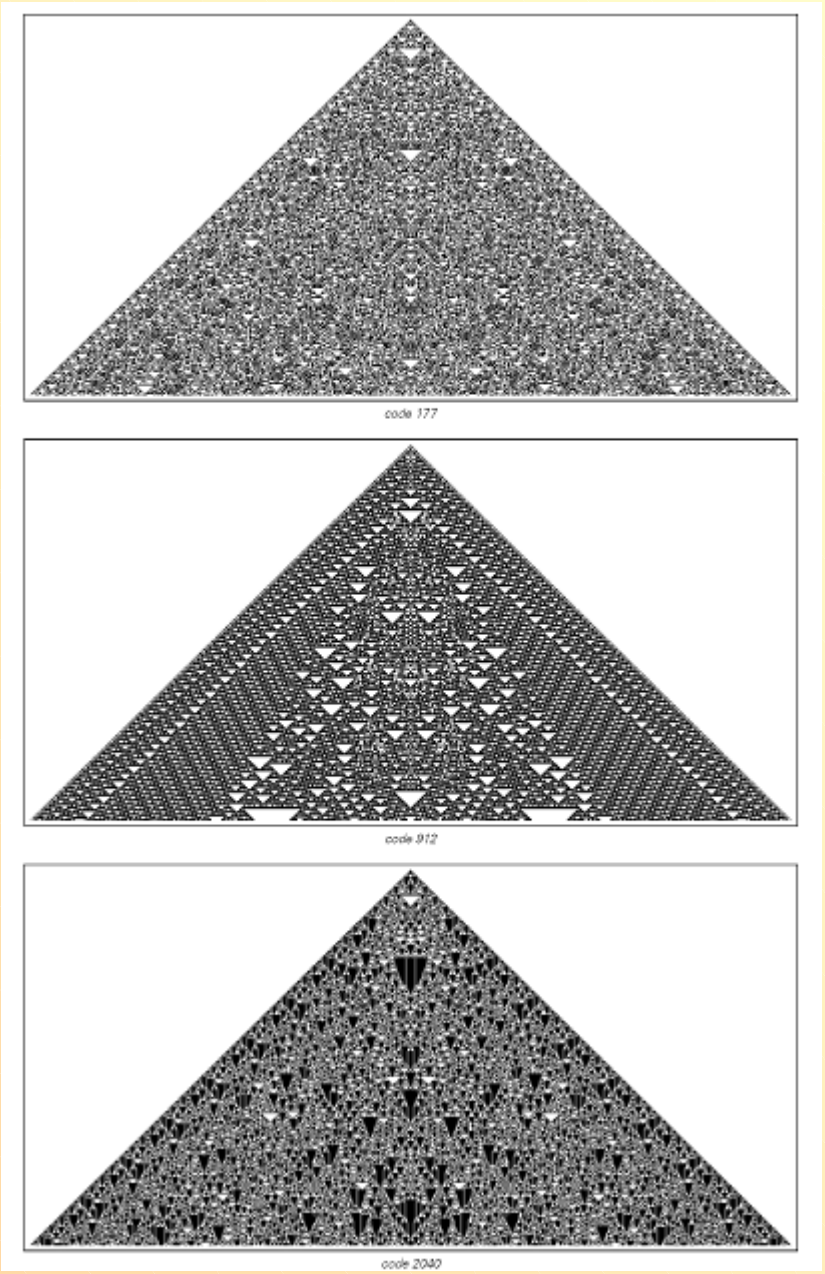
More Cellular Automata

- Results look similar to two-color automata.
- More complicated underlying rules do not seem to add to overall complexity.



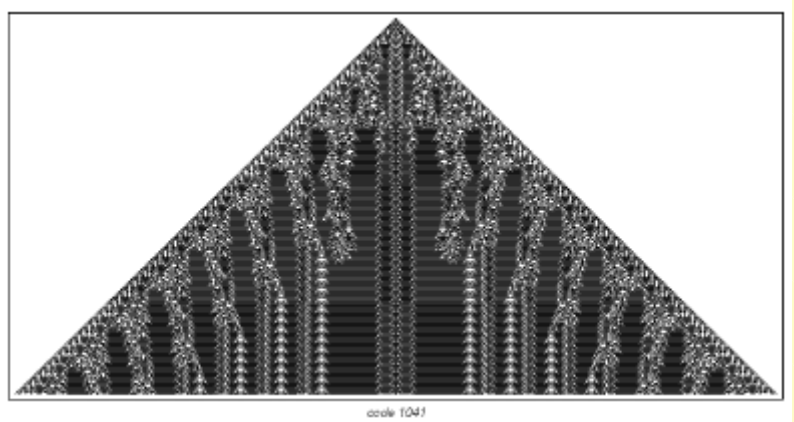
More Cellular Automata

- Three-color automata can still produce seemingly random results.
- Bilateral symmetry in these images is due to the basic structure of totalistic rules

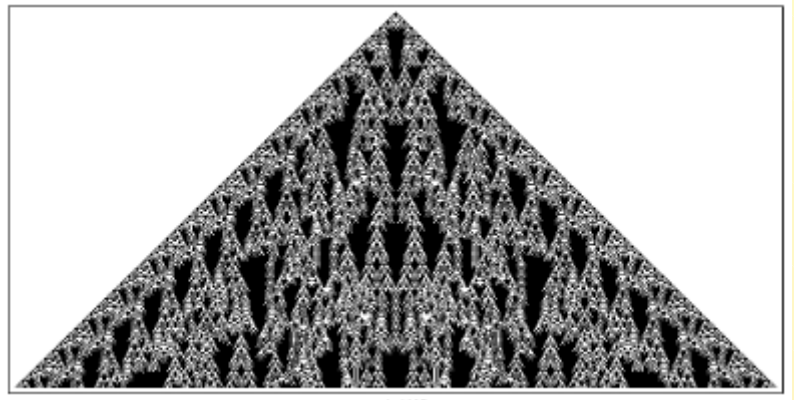


More Cellular Automata

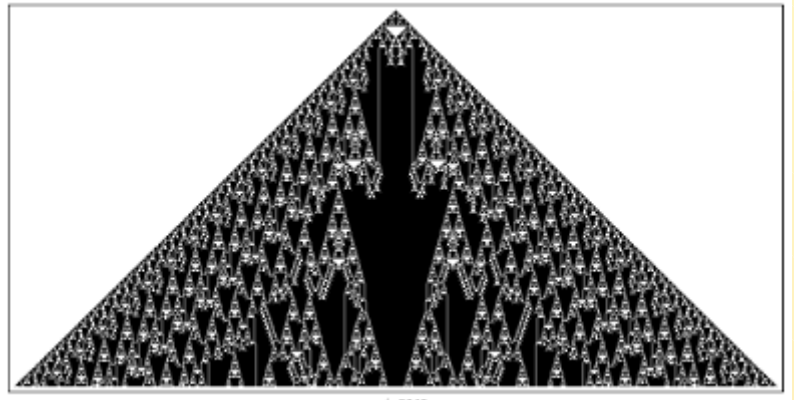
- 85% of 3-color automata produce behavior that is ultimately quite regular.
- Some rules, however, produce structures that have an interesting interaction between regularity and irregularity.



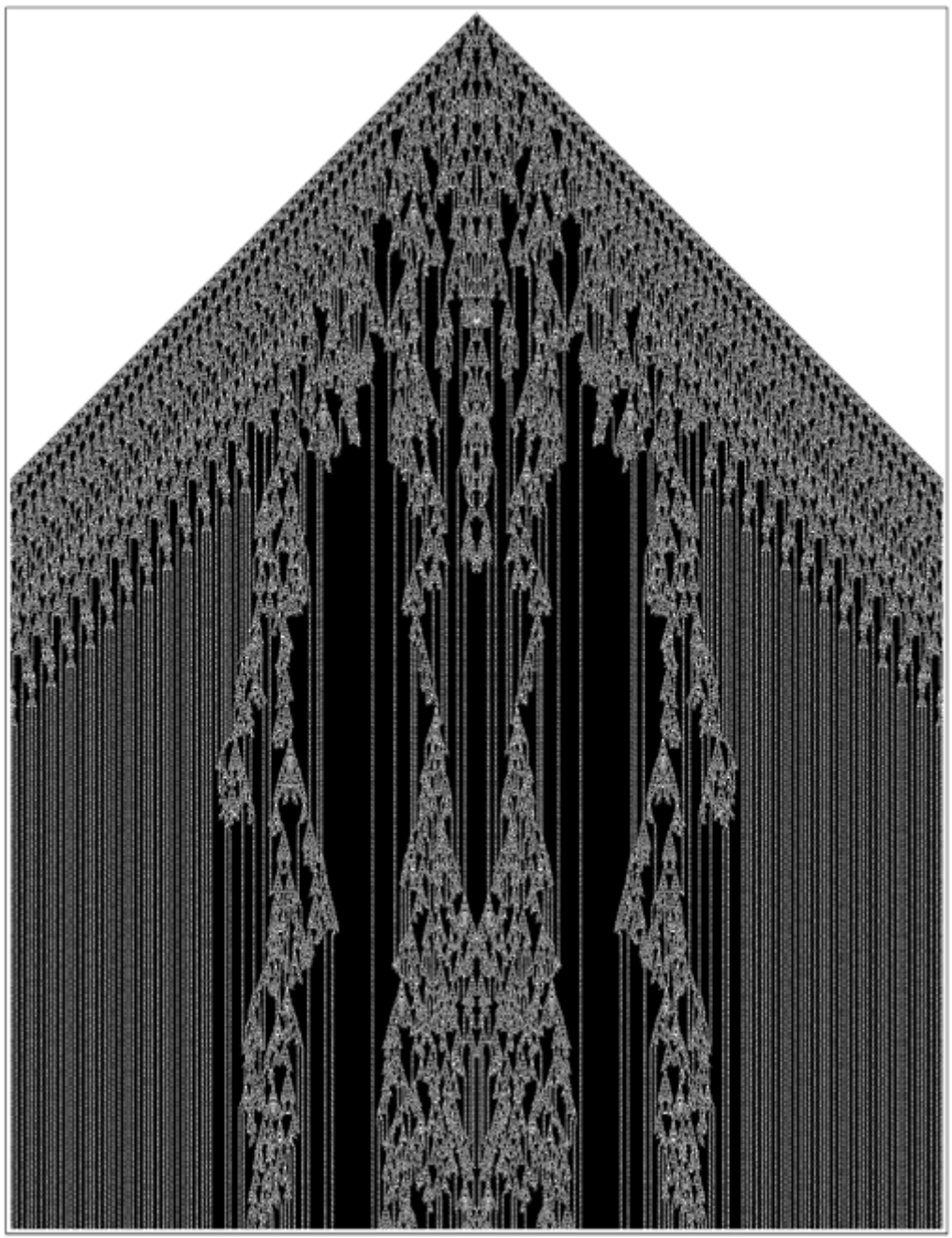
code 1041



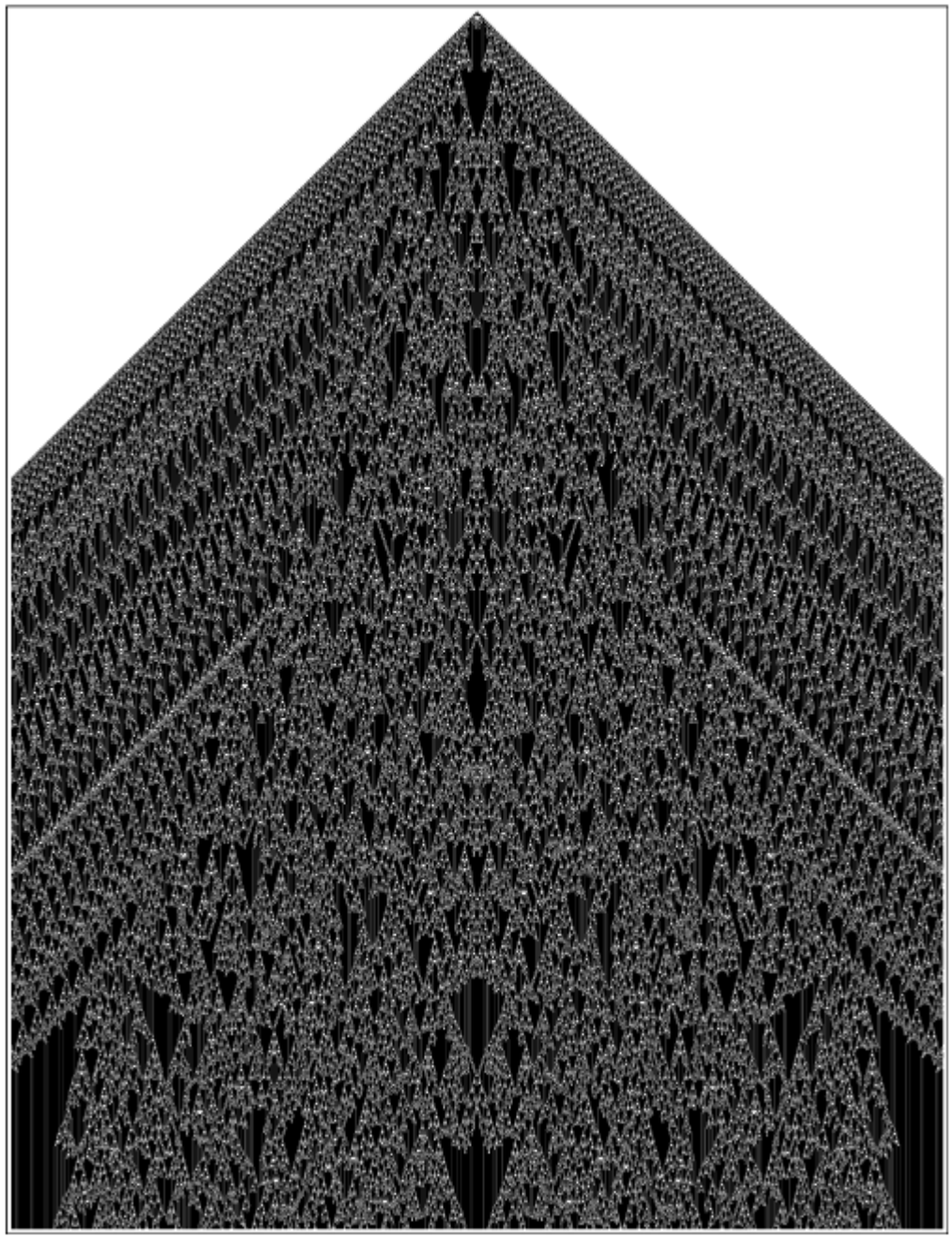
code 1635



code 2049



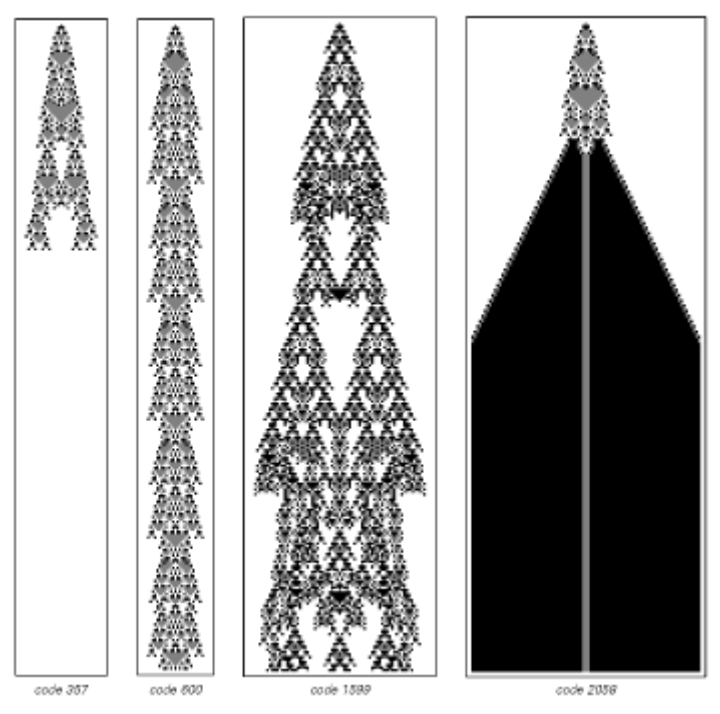
code 1636

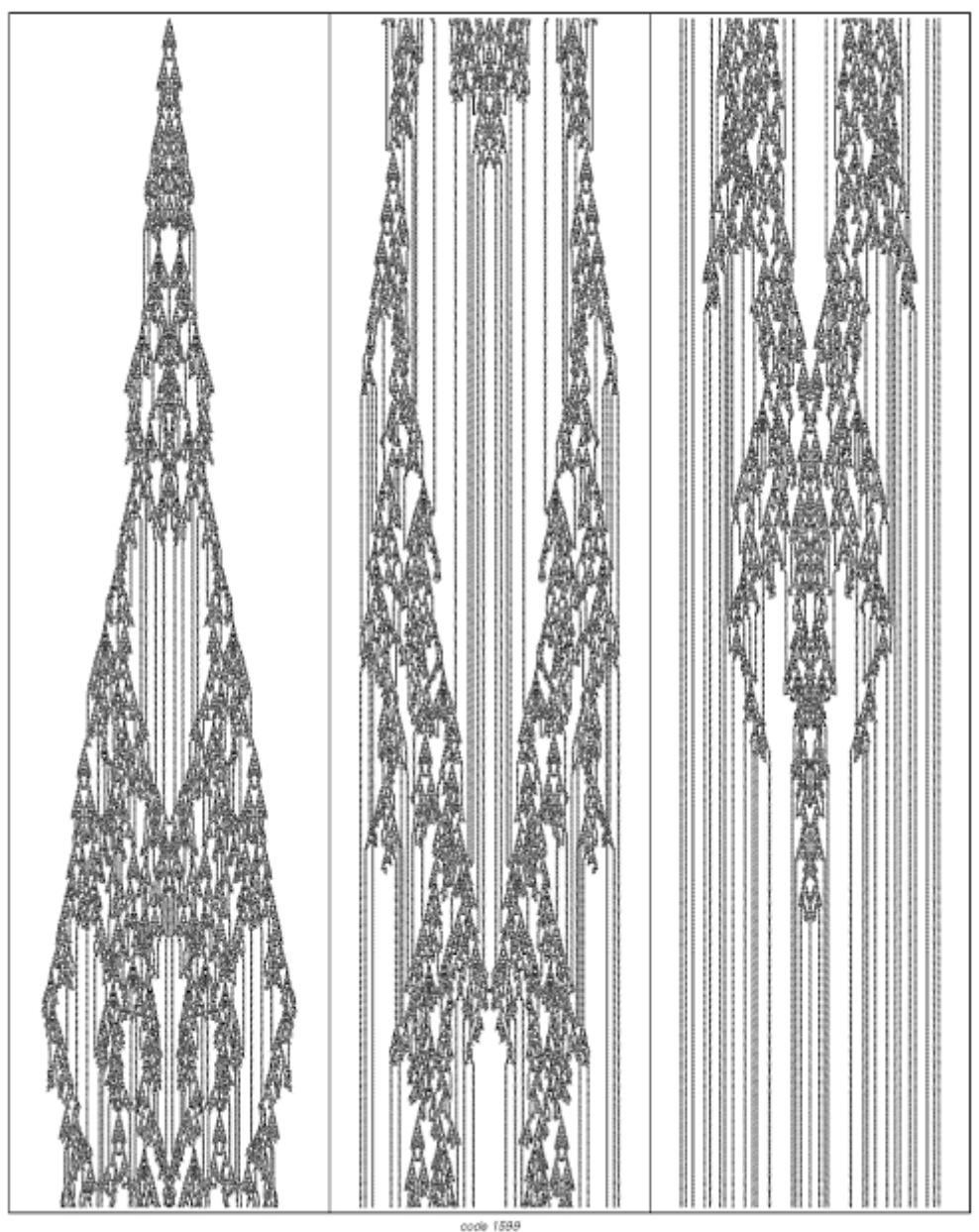


code 2049

More Cellular Automata

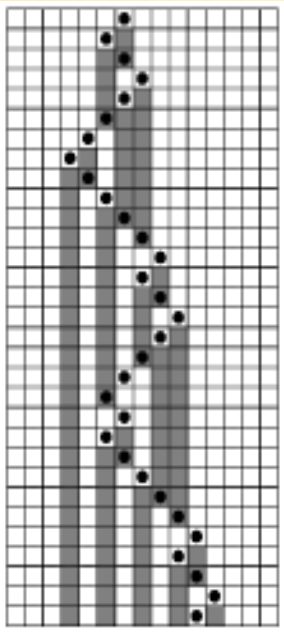
- These seem to be complex at first, but end up resolving to simple forms (even if it takes 8,282 steps).





Mobile Automata

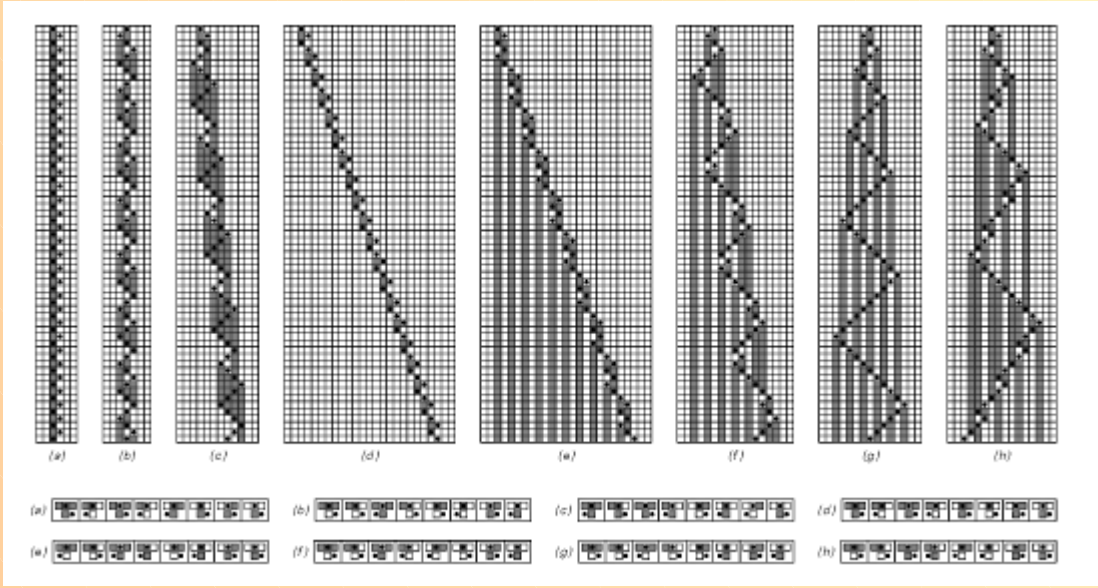
- Instead of updating all cells in parallel, there is only a single “active” cell that can change its color.
- Rules define how this cell can move and how it can change colors.



An example of a mobile automaton. Like a cellular automa
mobile automaton consists of a line of cells, with each cell
two possible colors. But unlike a cellular automaton, a
automaton has only one "active cell" (indicated here by a bla
at any particular step. The rule for the mobile automaton sp
both how the color of this active cell should be update
whether it should move to the left or right. The result of ev
for a larger number of steps with the particular rule shown
given as example (f) on the next page.

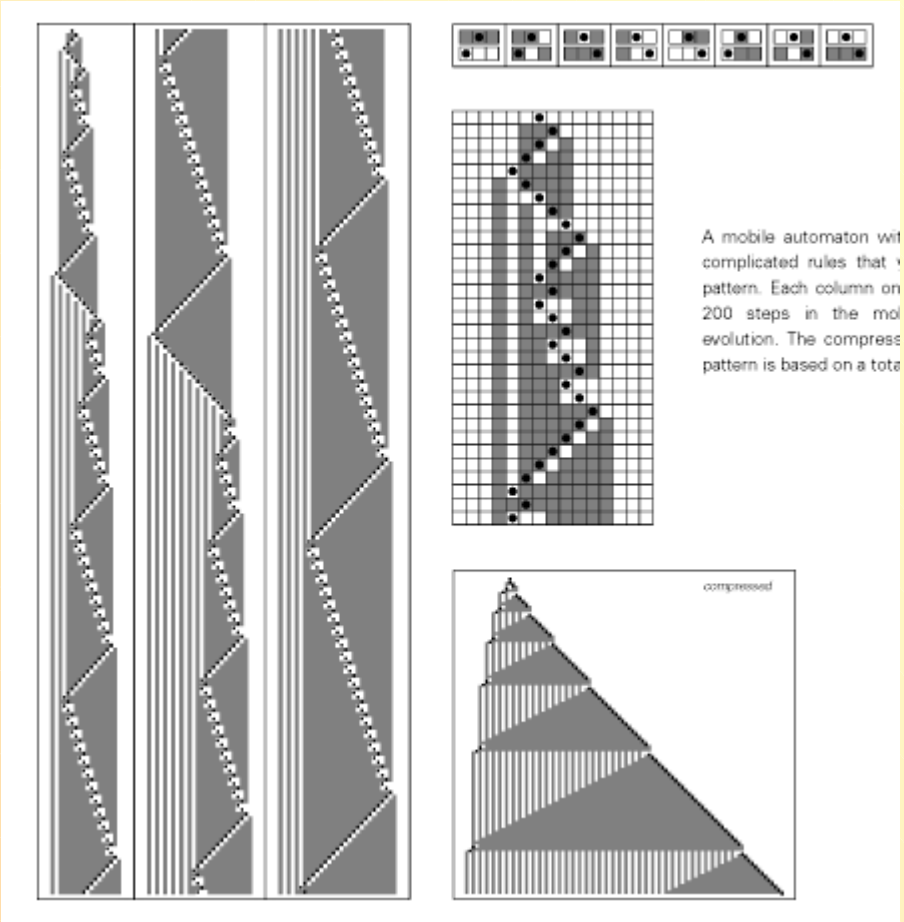
Mobile Automata

- 65,536 possible rule sets
- None *show the level of complexity of the cellular automata featured in the previous section.*



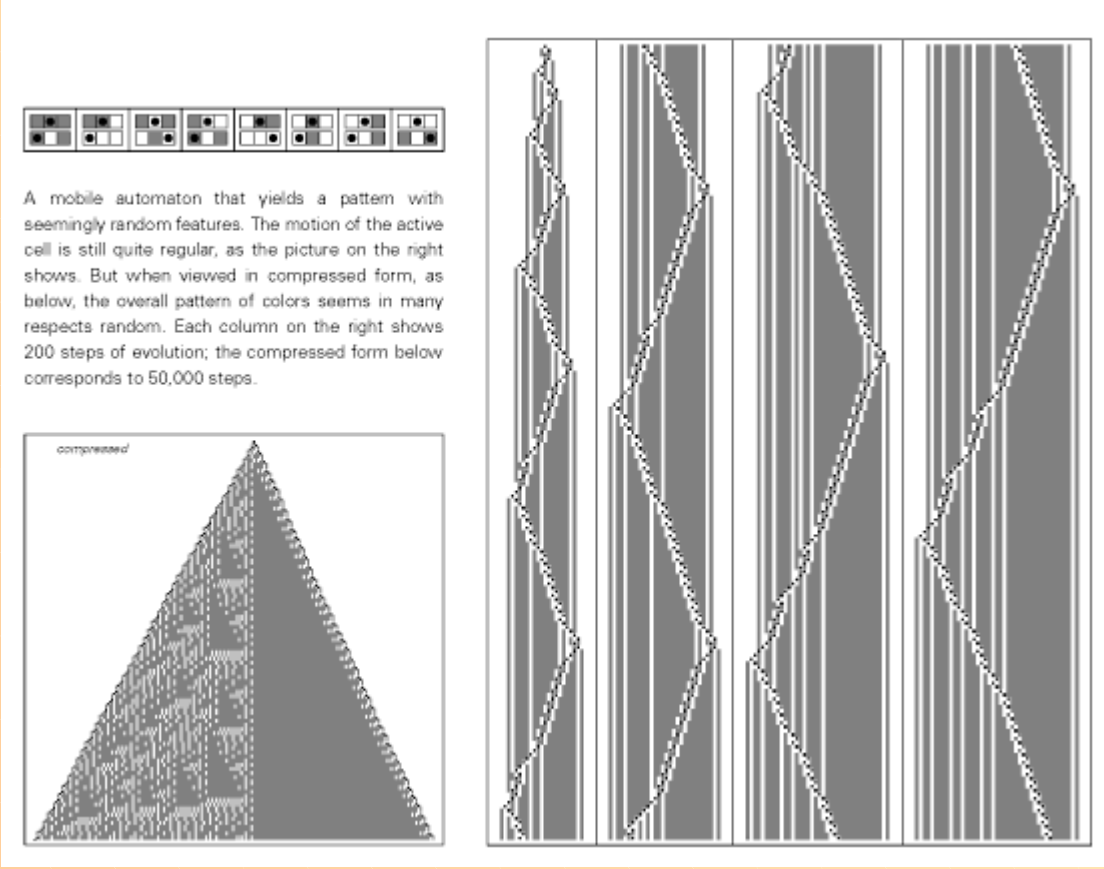
Mobile Automata

- Increase the complexity of the rules by allowing the active cell to change the colors of its neighbors as well, giving 4,294,967,296 possible rule sets.
- 99% show simple repetitive behavior
- Some, like the one to the right, create regular nested structures



Mobile Automata

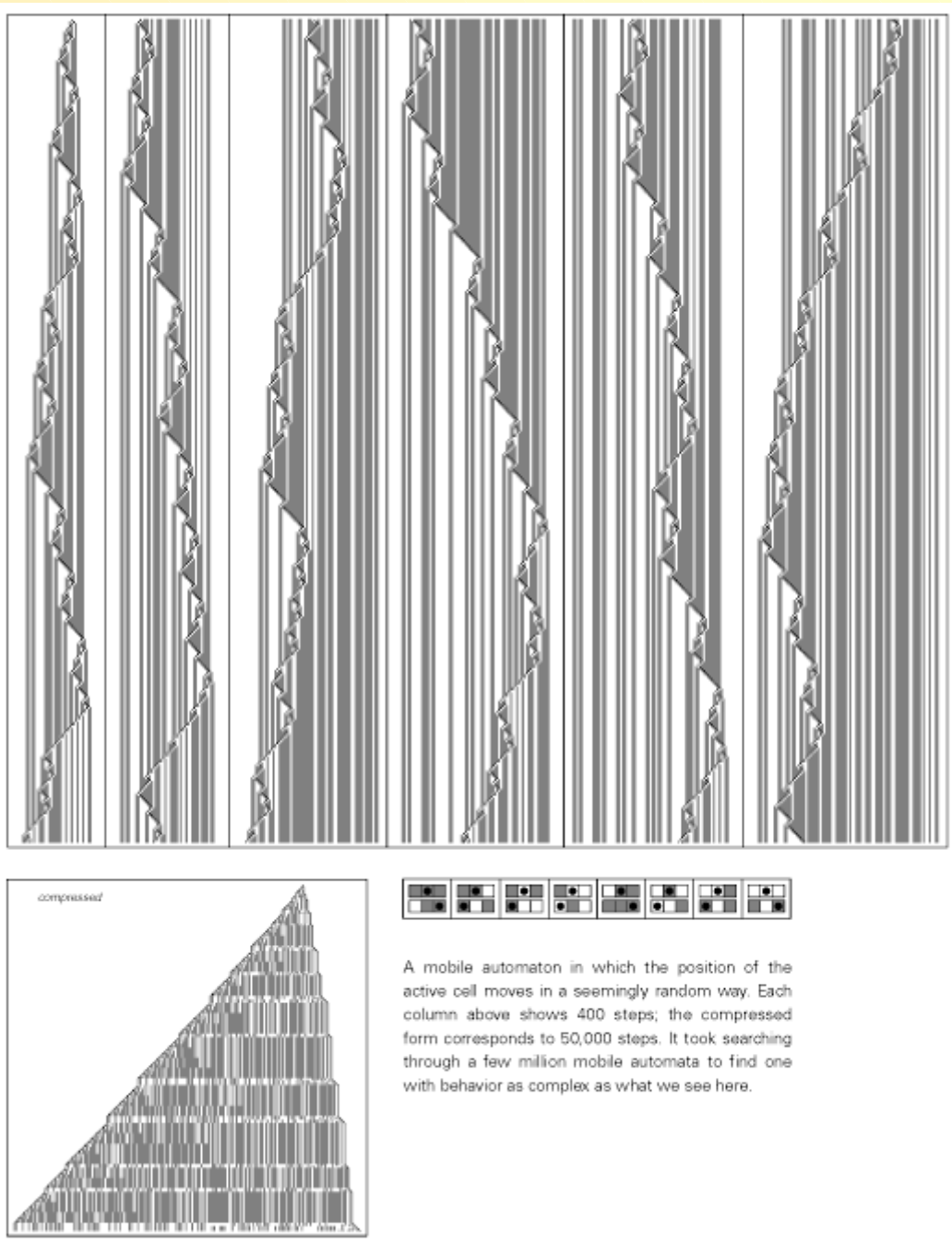
- About one in 50,000 rules give a seemingly random compressed form as below, though the movement of the active cell still seems somewhat regular.



A mobile automaton that yields a pattern with seemingly random features. The motion of the active cell is still quite regular, as the picture on the right shows. But when viewed in compressed form, as below, the overall pattern of colors seems in many respects random. Each column on the right shows 200 steps of evolution; the compressed form below corresponds to 50,000 steps.

Mobile Automata

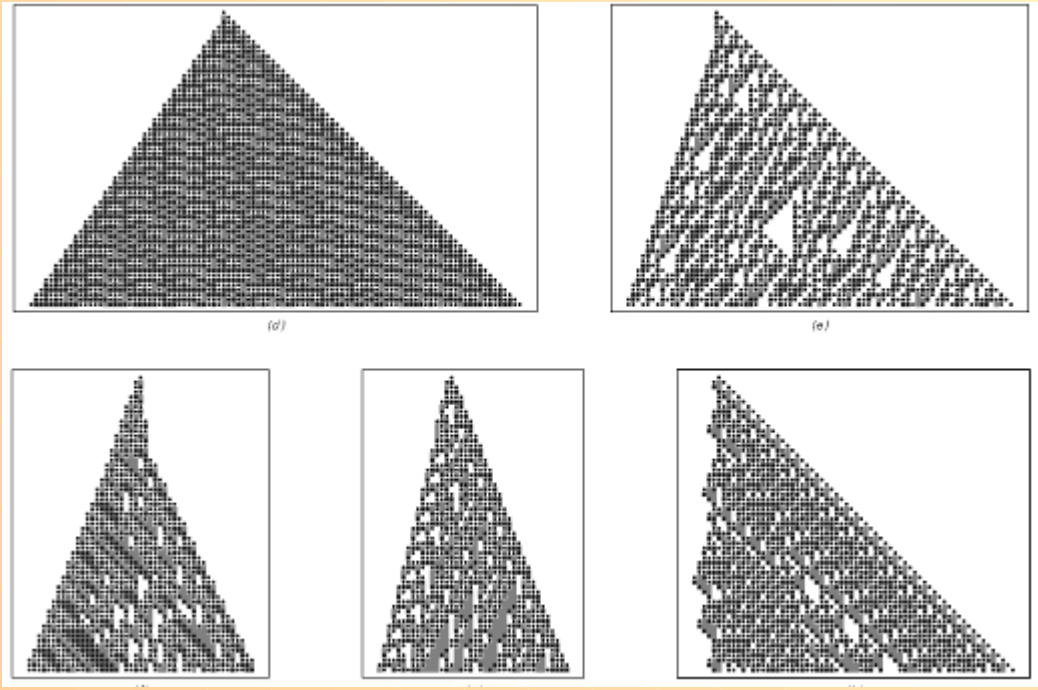
- One rule set in a “few million” will yield an image like the one seen here.



A mobile automaton in which the position of the active cell moves in a seemingly random way. Each column above shows 400 steps; the compressed form corresponds to 50,000 steps. It took searching through a few million mobile automata to find one with behavior as complex as what we see here.

Generalized Mobile Automata

- Mobile Automata where the active cell can move, split into two active cells, or disappear entirely.
- More complex behavior is relatively easier to find than with ordinary mobile automata.
- If every cell in a step becomes active, behavior is essentially that of a cellular automaton.

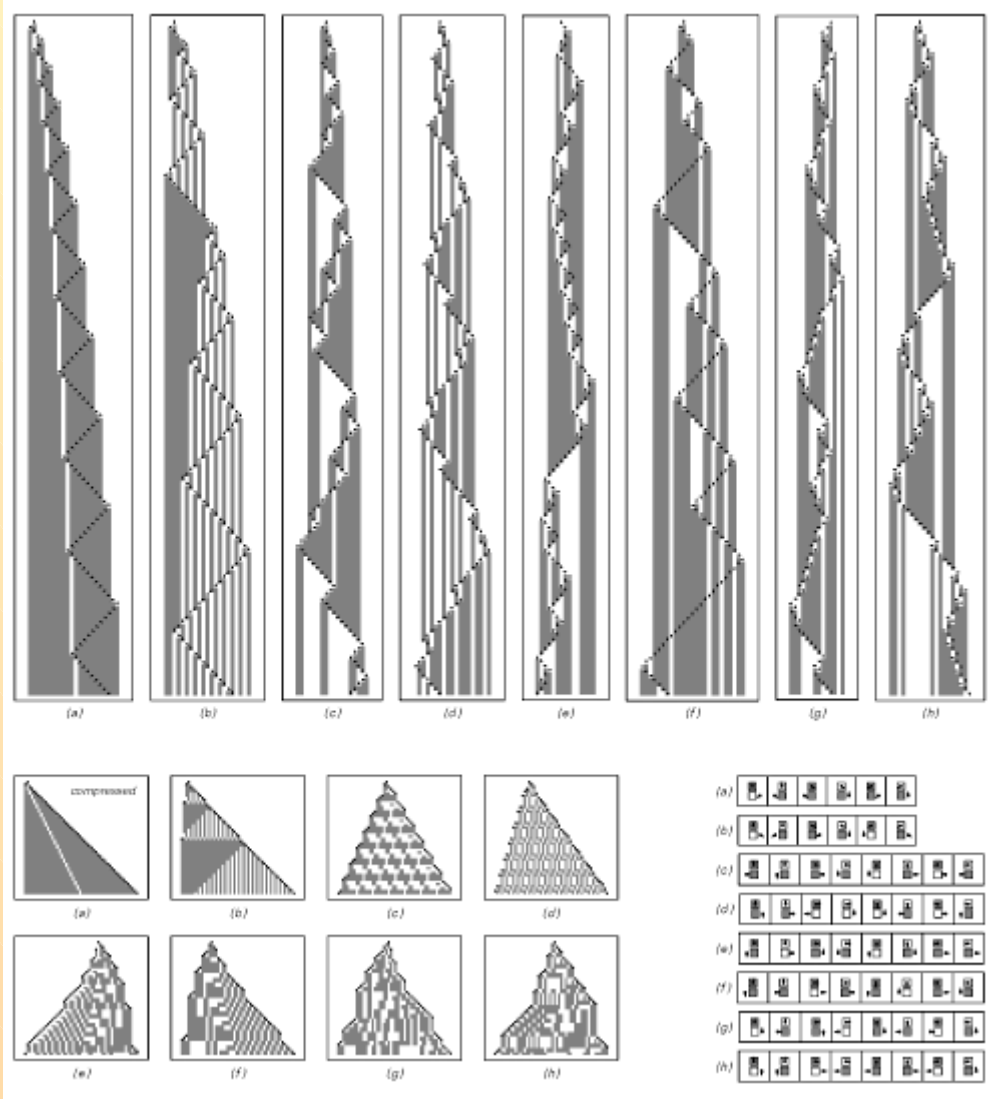


Turing Machines

- Like mobile automata: line of cells is a “tape,” active cell is the “head.”
- The head can have several possible states, and Turing rules can depend on the state of the head and the color of the cell at the head’s position, but not the color of neighboring cells.
- In a Turing Machine with two head states and two colors, none of the 4096 rule sets results in anything more complicated than repetitive or nested behavior.
- Likewise, with three head states, there is no apparent random behavior in any of the approximately 3 million possible machines.

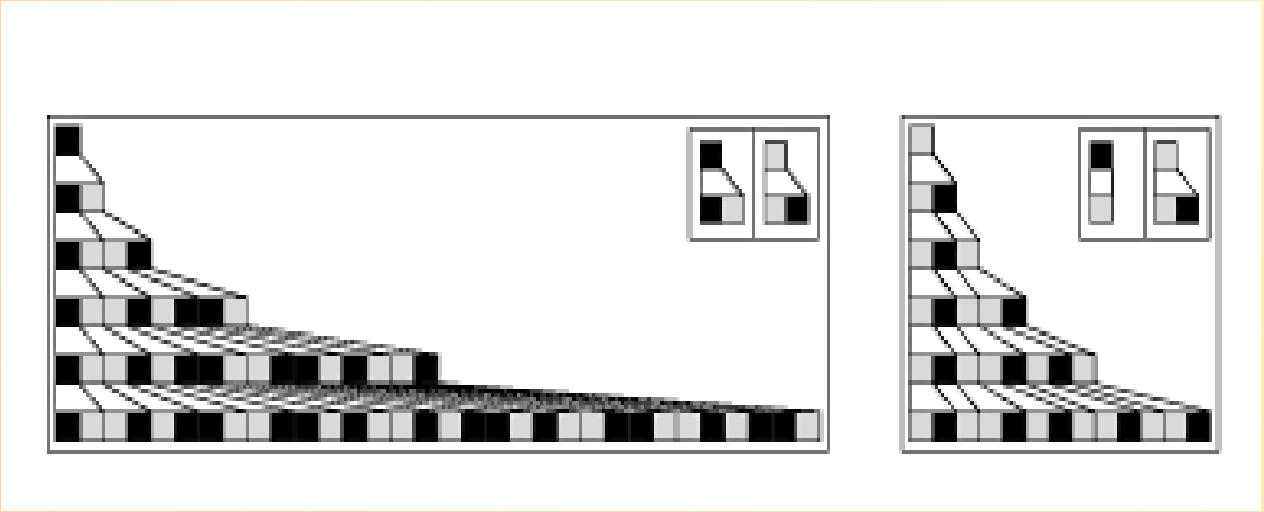
Turing Machines

- With four states for the head, five out of every million or so rules will have apparently random behavior.
- More than four states does not correspond to a significant increase in complexity.



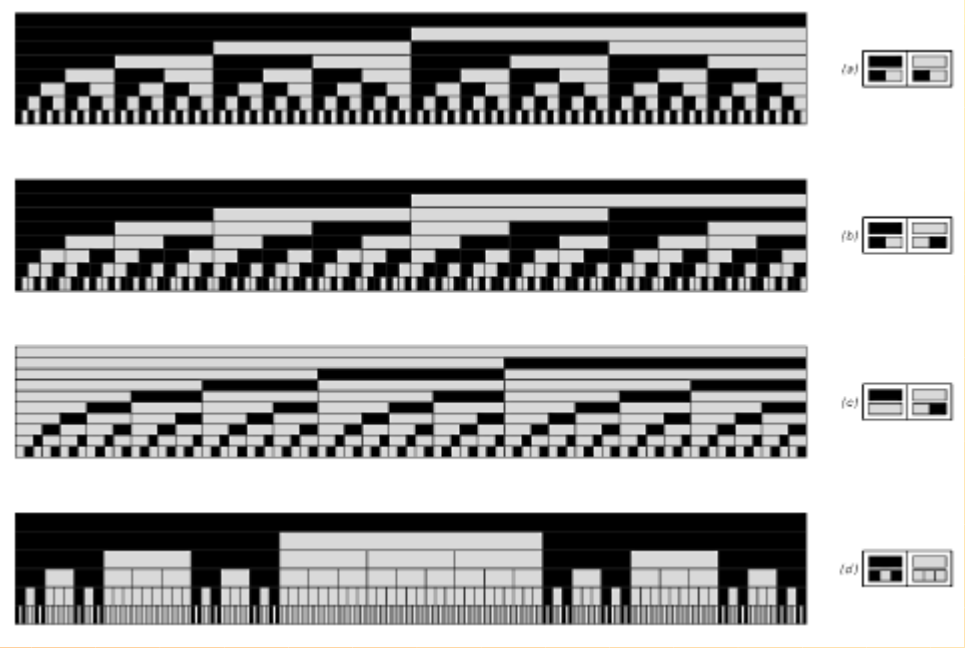
Substitution Systems

- Number of cells in an array can change
- Each cell is replaced with some sequence of cells depending on its color.
- You can also think of it as starting with a single large cell and subdividing it based on some rule.



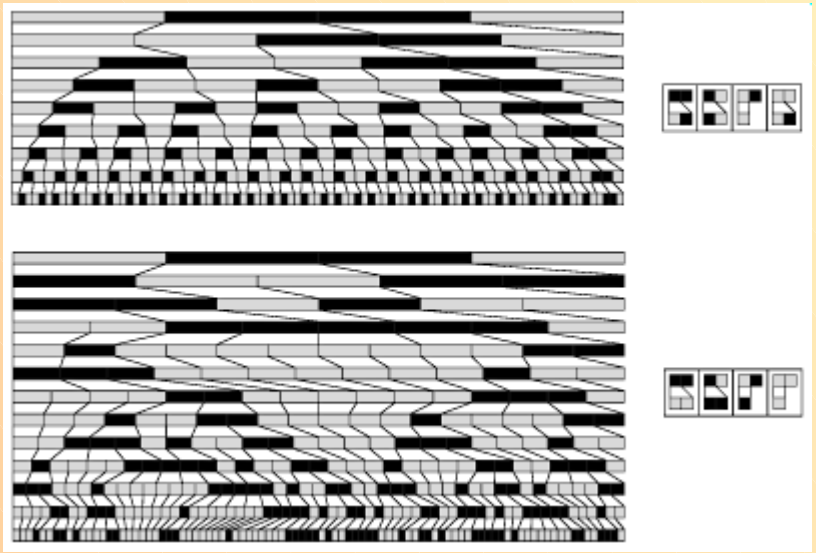
Substitution Systems

- Obvious regularity in all patterns produced by this method.
- Every time a cell of a specific color is encountered, it is split in the same way.



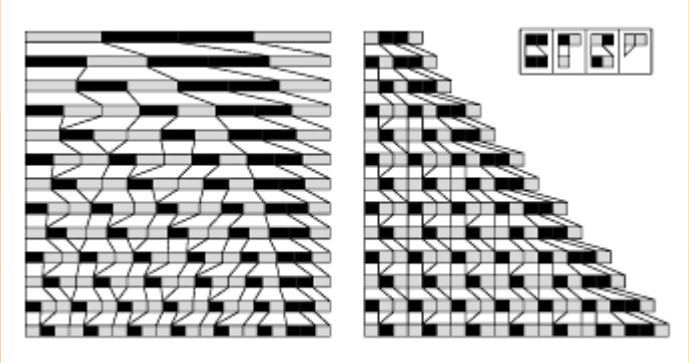
Substitution Systems

- Consider rules based not only on individual cells, but also on the neighbors of those cells.
- Can produce complex behavior
- Total number of elements never decreases from one step to the next.



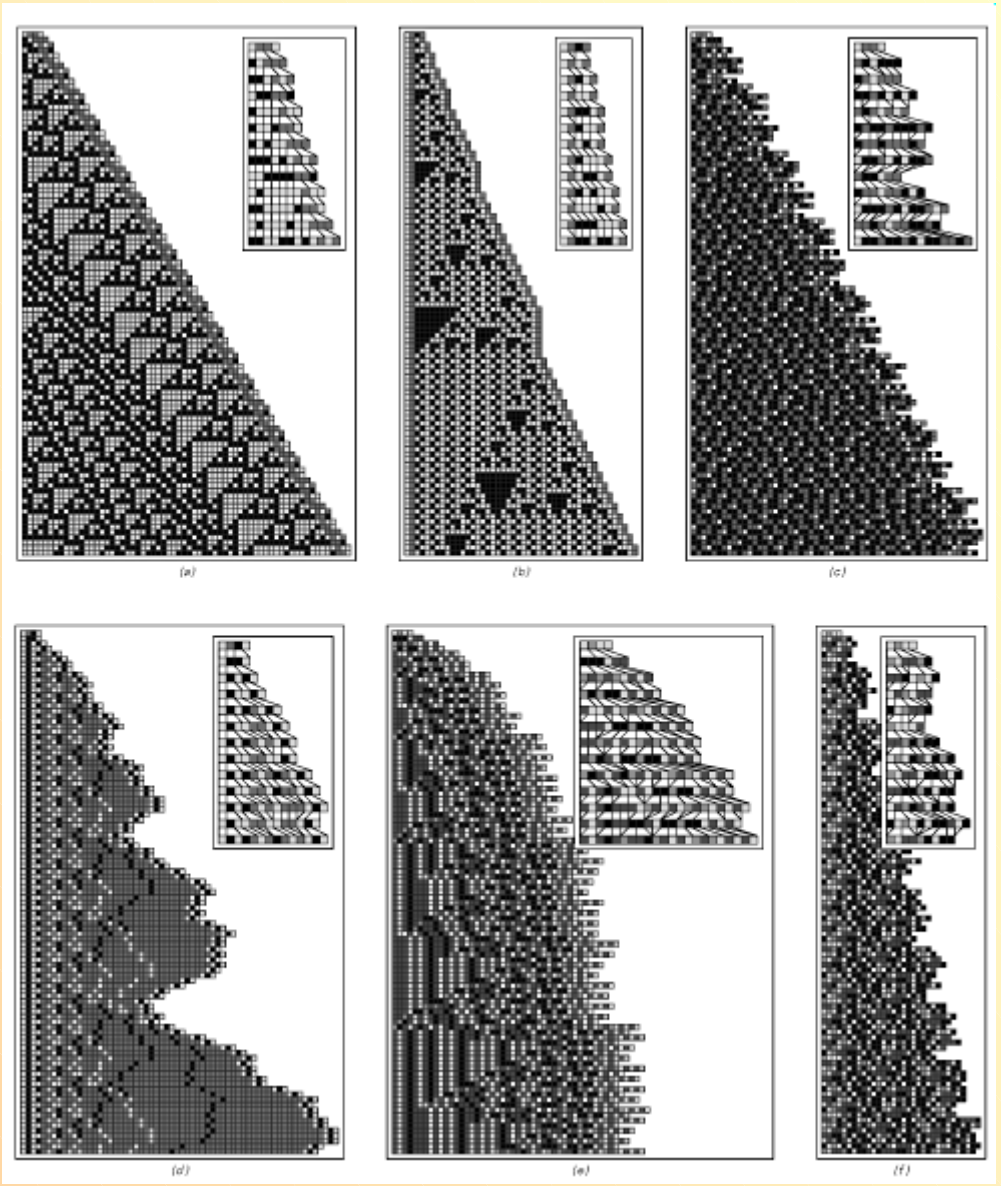
Substitution Systems

- It is also possible to have systems where elements can simply disappear.
- For a system to last, there needs to be a balance between elements created and elements destroyed at each time step.
- All such “slow growth” systems are simply repetitive.



Substitution Systems

- However, complex slow-growth systems can be achieved by increasing the number of available colors from 2 to 3 or 4.

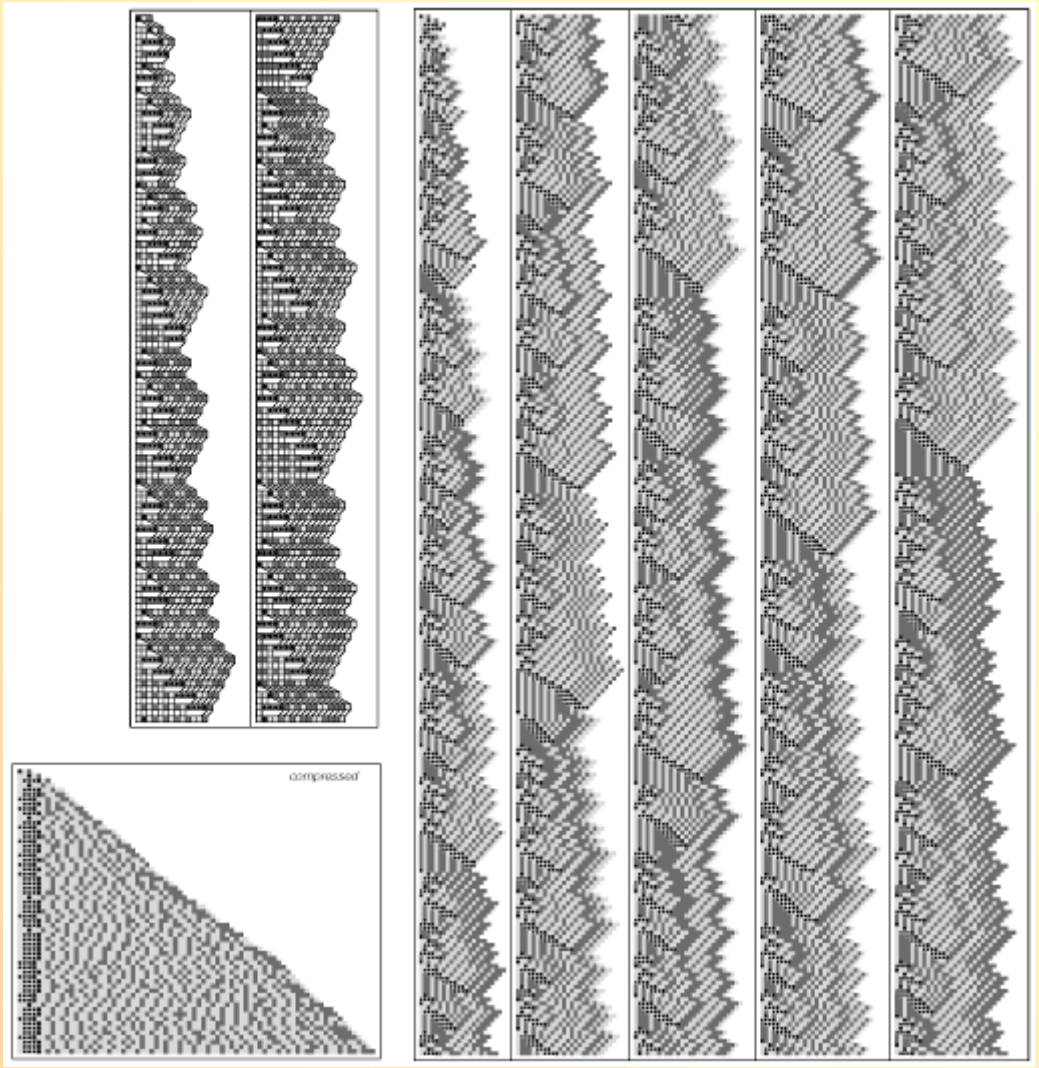


Sequential Substitution Systems

- Consider a substitution system that, instead of operating on all cells in parallel, replaces only the first instance of a particular string of cells.
- Analogous to a find-and-replace function of a text editor.
- If only the first or first and second instances of a string are replaced, no apparently complex behavior is observed.

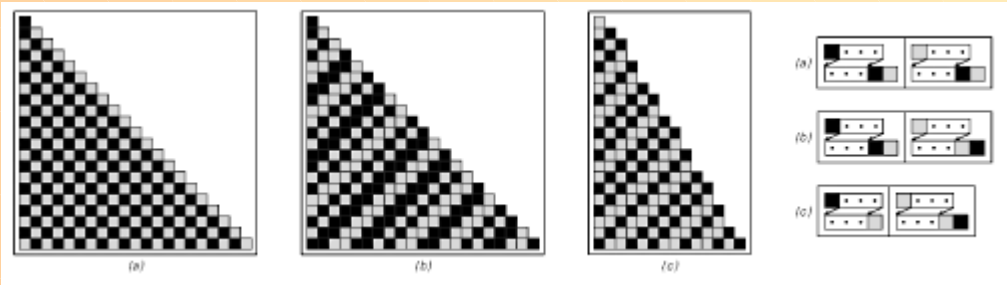
Sequential Substitution Systems

- However, if you allow more than 2 substitutions per step, more complex behavior is immediately observable.
- Apparently random patterns occur once every 10,000 or so rules.



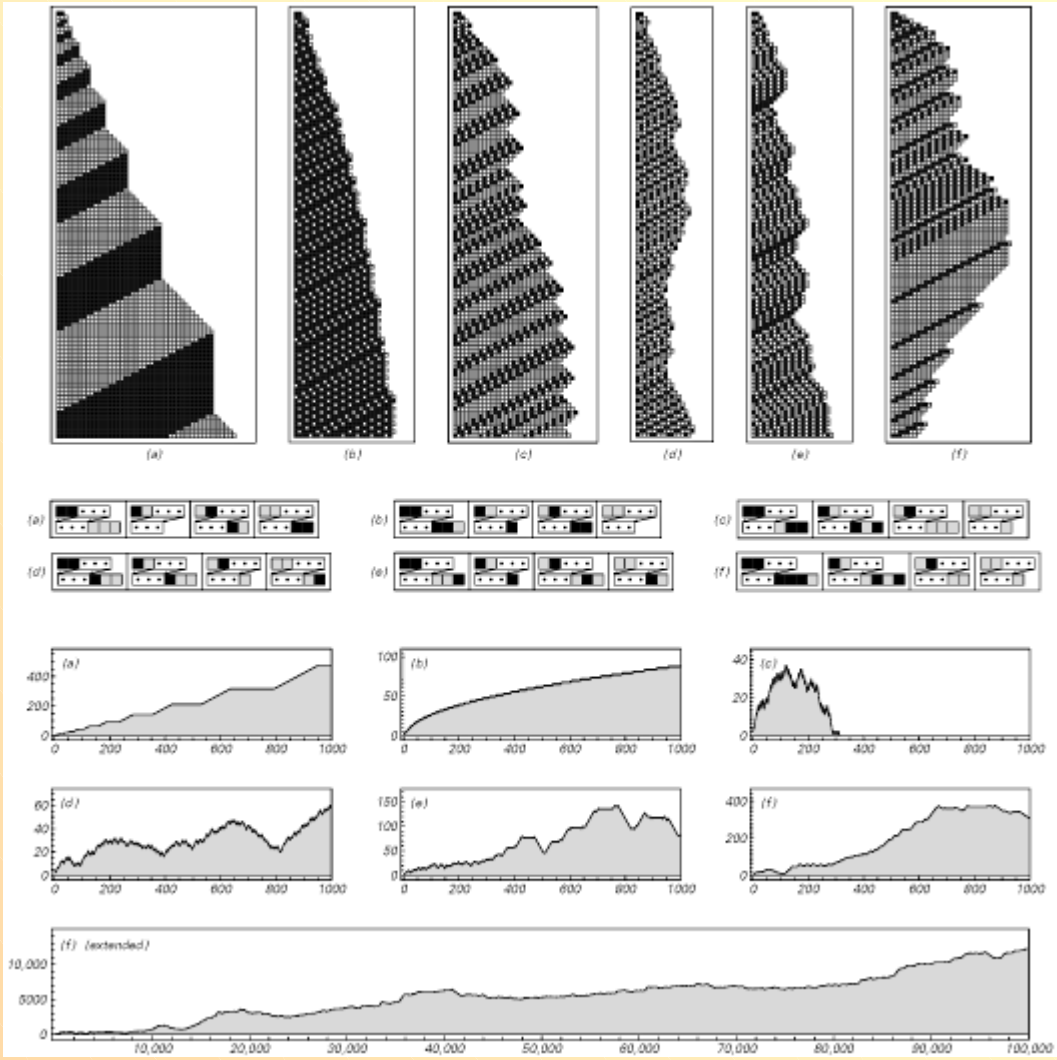
Tag Systems

- A Tag system consists of a sequence of elements, each colored, say, black or white.
- At each step, a fixed number of elements is removed from the beginning of the sequence.
- Depending on the color of the removed elements, one of several possible blocks is tagged onto the end of the sequence.



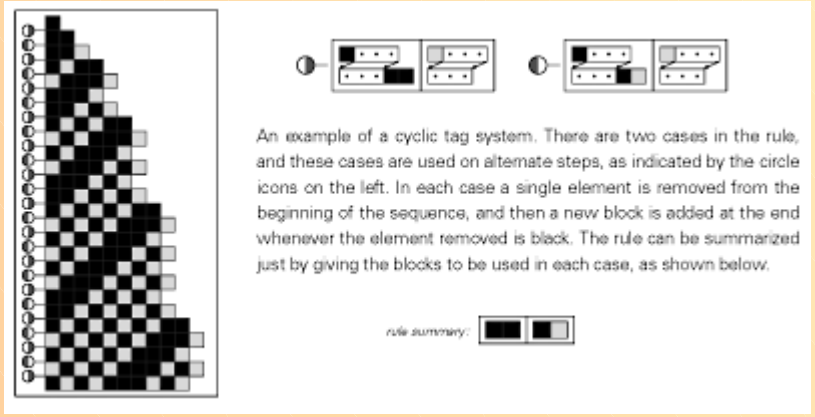
Tag Systems

- When only one element is removed, no complex behavior occurs (acts like a simple substitution system).
- However, when two elements are removed every step, more complex behavior appears.

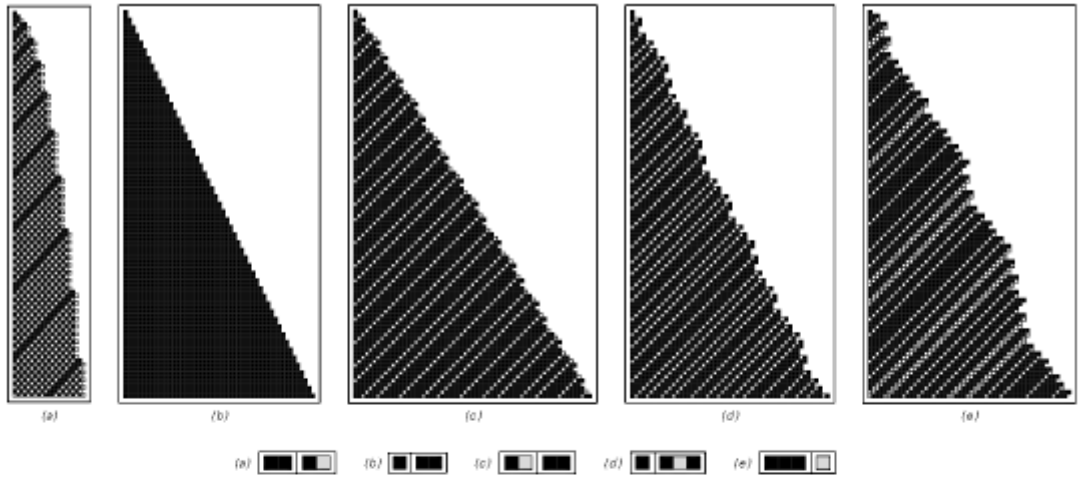


Cyclic Tag Systems

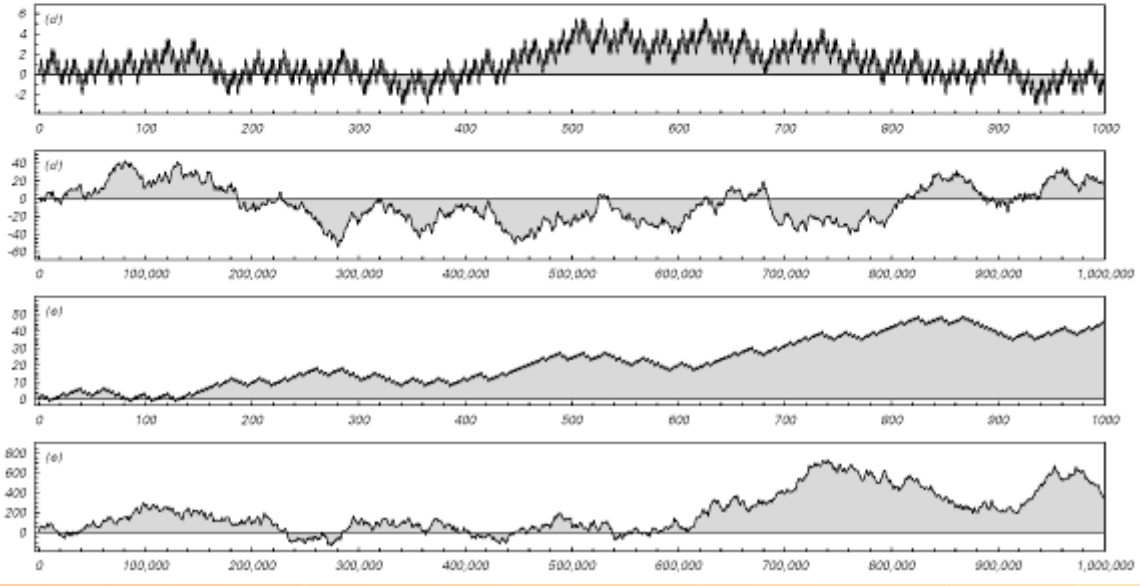
- Simple Case: two alternating rules, each of which tags a different block to the end of the sequence if the first element in the sequence is black.
- With more complex rules, one notices seemingly random fluctuations in length of the sequence.



Cyclic Tag Systems



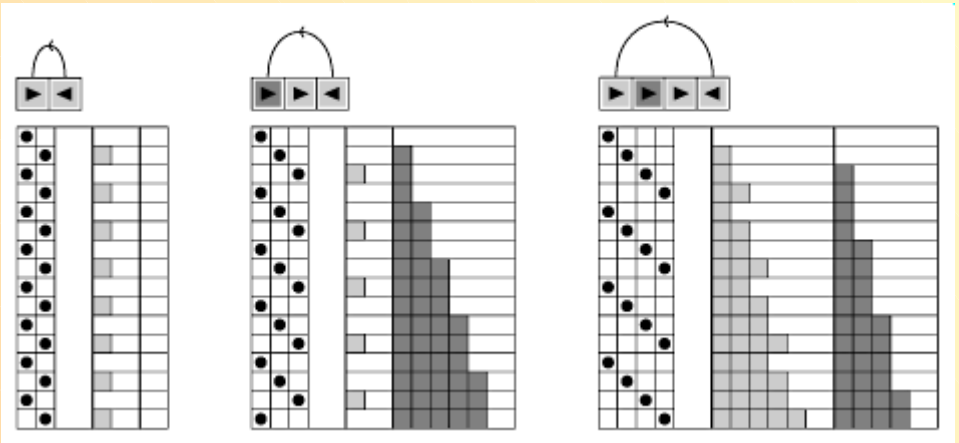
Examples of cyclic tag systems. In each case the initial condition consists of a single black element. In case (c), alternate steps in the leftmost column (which in all cyclic tag systems determines the overall behavior) have the same nested form as the third neighbor-independent substitution system shown on page 83.



Register Machines

- Specifically designed to work like simple idealized computers.
- At the lowest level, standard CPUs have registers that store numbers, and programs are converted into commands that specify operations to be performed on these registers.
- Increment commands simply add one to the number stored in a register.
- Decrement-jump commands decrease the value by one and jump to some other point in the program, where execution continues.
- The number in a register is assumed to be non-negative, so when a decrement-jump is applied to a zero register, nothing happens, and execution continues with the next command in the program.

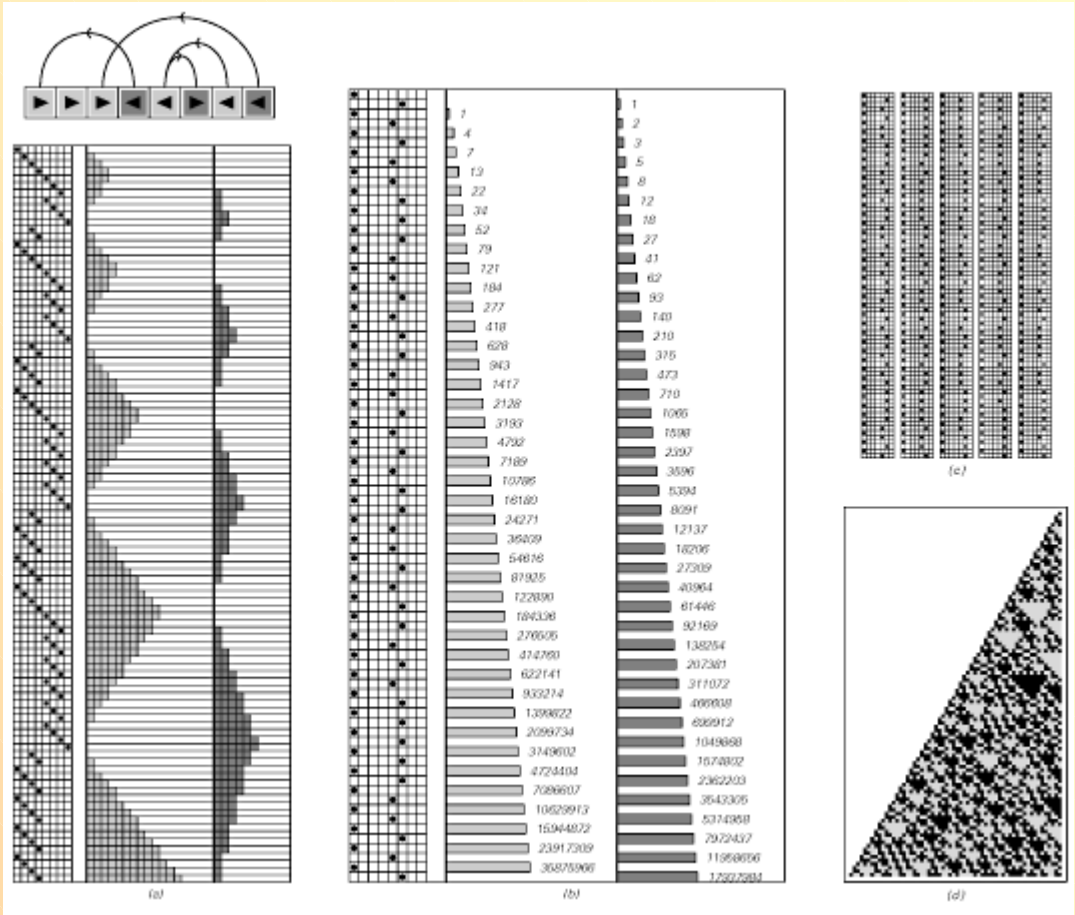
Register Machines



- All of the 10,552 possible machines with 4 or fewer instructions are essentially repetitive.
- One of the 276,224376 machines with 5 instructions displays a nested structure, but none are more complex.
- 126 of the 11,019,960,576 machines with 8 instructions show more complicated behavior.

Register Machines

- Part (a) is the ordinary evolution.
- Part (b) is compressed to show only the steps where one of the registers has decreased to zero.
- Part (c) shows the instructions executed the first 400 times that one of the registers decreases to zero.
- Part (d) shows the successive values (in binary digits) obtained by the second register at steps where the first register has decreased to zero.

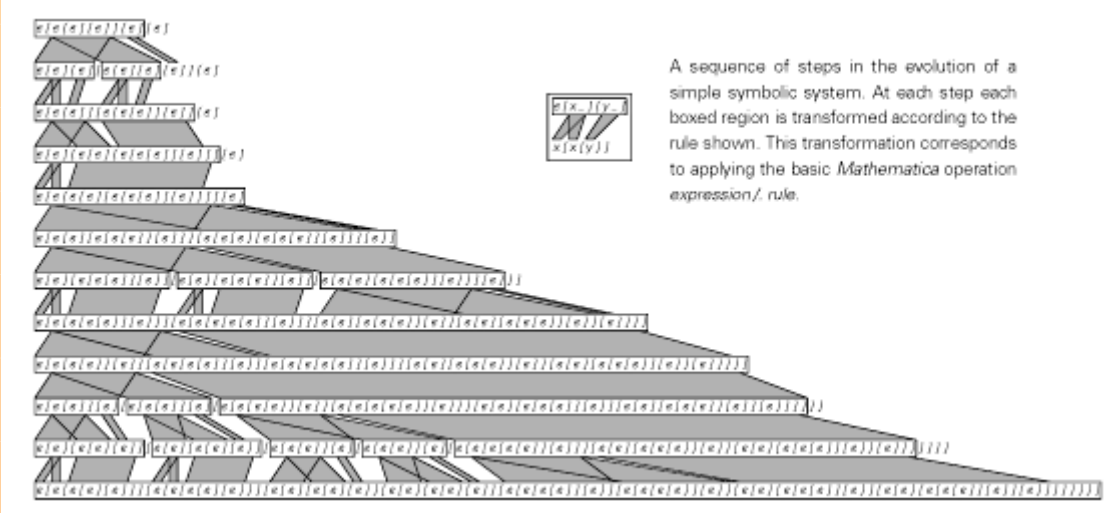


Register Machines

- No significant gain in complexity with more than 2 registers.
- Extending rules to allow for addition, subtraction, or comparison, while not more complex, allow for more accurate idealizations of low-level computer operations.
- Results obtained here can be generalized to apply to programs written in C, Java, Basic, or assembler.

Symbolic Systems

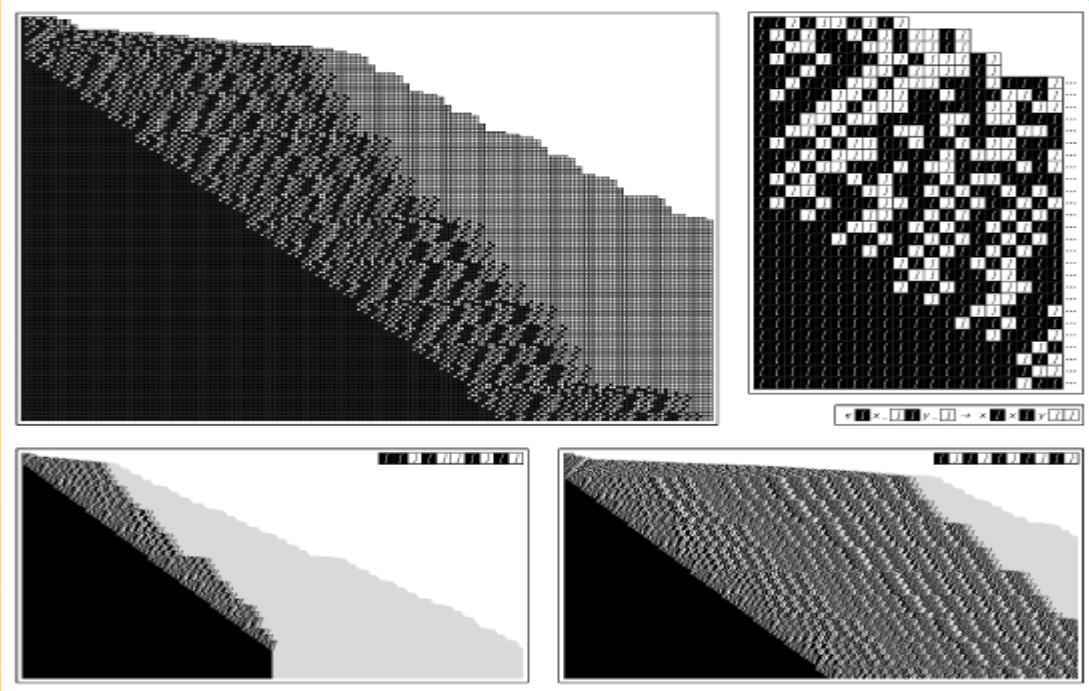
- Consider a mathematical expression with nested parentheses, like e[e[e][e]][e][e].
- Using transformation rules like e[x_][y_]->x[x[y]] (where x_ and y_ can be any expression), apply the rule from left to right wherever possible without overlapping.



A sequence of steps in the evolution of a simple symbolic system. At each step each boxed region is transformed according to the rule shown. This transformation corresponds to applying the basic *Mathematica* operation *expression*/. *rule*.

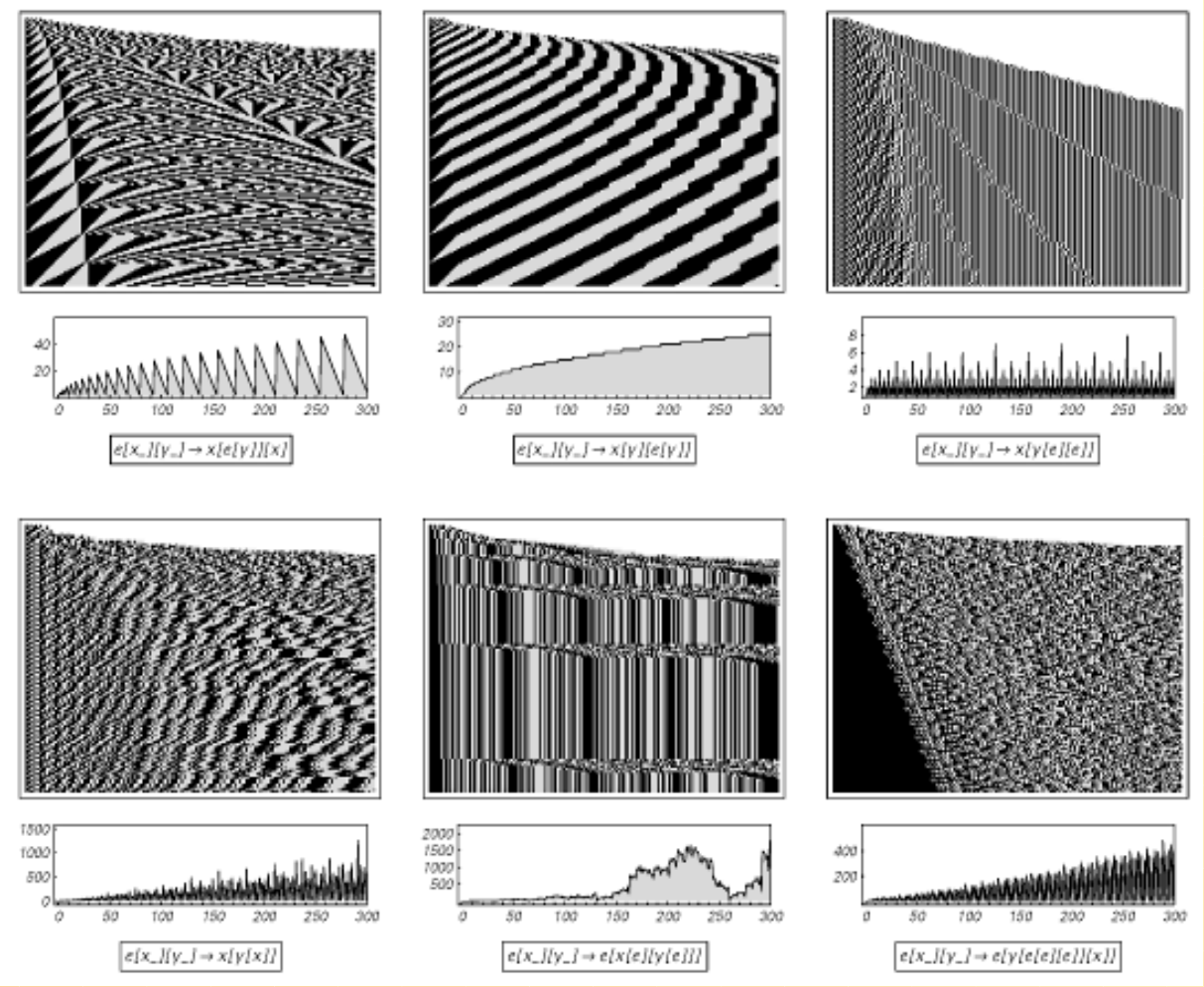
Symbolic Systems

- Pictures obtained by representing opening and closing brackets as dark and light squares, respectively.
- These use the same rule as the previous slide, with different initial conditions.
- All these will stabilize, sometimes after a very long time.



Symbolic Systems

- Plots of the same initial condition, but with different rules applied.



Some Conclusions

- What characteristics of cellular automata are responsible for their tendency for complexity?
- From substitution systems, we see it is not necessary to have a rigid grid of elements.
- From mobile automata, we see that updating in parallel is not necessary.
- Each system type discussed was chosen to take away features from the cellular automata model, but all are ultimately capable of complexity.

Some Conclusions

- When, in general, does complexity occur?
  - Extremely simple rules lead to repetitive behavior.
  - Slightly more complicated rules lead to nesting.
  - Once some “complexity threshold” is passed, randomness appears, but does not seem to become more likely as rules are made even more complex.
  - In general, this threshold is very low.

Some Conclusions

- Repetition, nesting, and apparent randomness are general concepts that can be applied to a wide range of systems.
- Therefore, even if the details of a system are unknown, we can still potentially make fundamental statements about its overall behavior.

How the discoveries in this chapter were made

- Methodology based on doing computer experiments.
  - Initialize a system with state and rules, and observe resultant behavior.
  - Allows one to discover new, unexpected phenomena
  - Perfectly repeatable anywhere, at any time.
- By observing systems with the simplest possible structure, you can get results with broad and fundamental significance.
- Simple systems are easy to implement on a computer and investigate systematically.

How the discoveries in this chapter were made

- In the author's view, the single most common mistake in doing computer experiments is making them more complicated than is necessary, especially when such simple systems can lead to apparent complexity.
- “One can never start with too simple a system.”

How the discoveries in this chapter were made

- Do a mindless search of a large number of cases than a carefully crafted search of a smaller number.
- Look explicitly at actual behavior instead of a summary.
- To whit, display the behavior as a picture or diagram that is easily and quickly processed by the naked eye.
- Avoid assumptions and set up experiments that are simple and direct enough not to miss important new phenomena.