CELLULAR AUTOMATA AND APPLICATIONS

GAVIN ANDREWS

1. Introduction

This paper is a study of cellular automata as computational programs and their remarkable ability to create complex behavior from simple rules. We examine a number of these simple programs in order to draw conclusions about the nature of complexity seen in the world and discuss the potential of using such programs for the purposes of modeling. The information presented within is in large part the work of mathematician Stephen Wolfram, as presented in his book *A New Kind of Science*[1].

Section 2 begins by introducing one-dimensional cellular automata and the four classifications of behavior that they exhibit. In sections 3 and 4 the concept of computational universality discovered by Alan Turing in the original Turing machine is introduced and shown to be present in various cellular automata that demonstrate Class IV behavior. The idea of computational complexity as it pertains to universality and its implications for modern science are then examined. In section

5 we discuss the challenges and advantages of modeling with cellular automata, and give several examples of current models.

2. Cellular Automata and Classifications of Complexity

The one-dimensional cellular automaton exists on an infinite horizontal array of cells. For the purposes of this section we will look at the one-dimensional cellular automata (c.a.) with square cells that are limited to only two possible states per cell: white and black. The c.a.'s rules determine how the infinite arrangement of black and white cells will be updated from time step to time step. Again, for the purposes of this section, we will look at c.a.'s whose rules are updated based on a nearest neighbor scheme. This means that to determine the state of a cell in position p at time step $t+1$, we will look at the states of cells in position $p-1$, p, and $p+1$ all in time step t. For each of the eight possible patterns of white and black cells, the state of cell p at time step $t+1$ is chosen as either black or white. See Figure 1 for the eight possible input patterns, as well as one possible output. In all there are 256 different possible outputs.

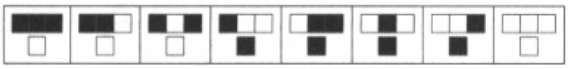


FIGURE 1. Along the top are the eight possible patterns of three, two-state cells. They are displayed in the conventional left to right order in this figure. The bottom is one possible set of outputs. In all, there are 256 different possible outputs. This image is scanned from Wolfram [1], page 53.

To analyze the behavior of these programs Wolfram developed a naming convention, a standard initial condition and method of viewing the results of multiple iterations at once. To name the basic one-dimensional c.a.'s described above, a hierarchy was given to eight possible patterns with black-black-black on the far left and white-white-white one the far right. Each combination was made to represent a place in the binary numbering system. Black-black-black for example was represents the 2^7th place. Assigning the value of zero to white and one to black gives each of the possible arrangements of updating scenarios a binary number that ranges from 0 to 255 in base ten. See Figure 2 for several examples of this naming scheme.

The initial conditions for all the rules, 0-255 consist of one black cell with rays of white cells extending to infinity on both sides. To view multiple iterations at once, each time step is placed below the previous one with the positions of each cell unchanging, for example see Figure 3. This creates a two-dimensional image of multiple iterations of a c.a.

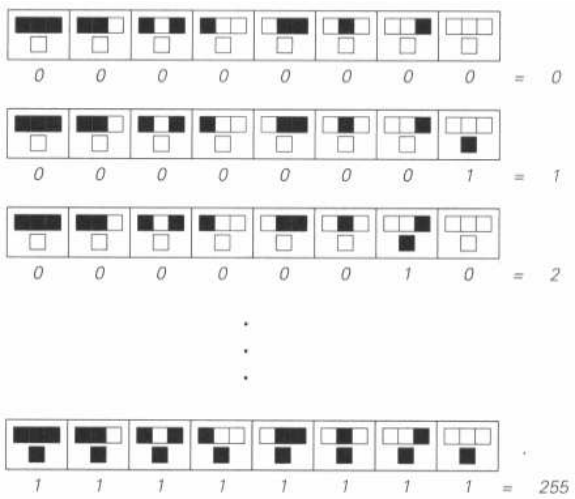
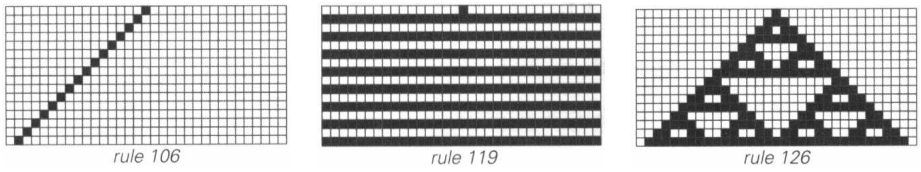


FIGURE 2. These are the first three rules and the last one. The sequence of zeros and ones are binary numbers with their base ten equivalent labeled to the right of each sequence. Wolfram, page 53.

allowing for analysis of behavior. It should be noted that the maximum rate of travel of the black square in the middle is one lateral square per iteration.

Wolfram classifies the behavior observed in these c.a.'s in four distinct classes. The first is Class I, which contains simple repeating behavior. This can range from a single vertical or diagonal line (the initial conditions remain) as in rules 100 and rule 106, or a series of alternating all white and all black iterations as in rules 119 and 21. See Figure 3, images (a) and (b) of rule 106 and 119 respectively for examples of Class I behavior. The behavior is easily recognizable as



(a) (b) (c)

FIGURE 3. Above are 16 iterations of rules 106, 119, and 126. Rule 106 and 119 are examples of Class I behavior, and 126 is one of Class II behavior. Wolfram, page 54.

containing repetitive elements of equal size that encompass the whole program. Approximately 86% of the 256 basic c.a.'s are of this class.

The second class of behavior, Class II, is characterized by c.a's that have nested patterns. A nested pattern is a configuration that repeats itself on an ever increasing scale. That is, smaller scale representations of a selected region occur within the region itself. Approximately 9% of the basic c.a.'s are of this class. See Figure 3, (c) for an image of a Class II nested pattern in rule 126.

Class III behavior is completely random. The c.a.'s in this class have shapes that repeat themselves, but their location and frequency is random. This class contains about 4% of the basic c.a.'s.

The final class, Class IV behavior, is a combination of Class I behavior and Class III behavior. These c.a.'s exhibit a complex combination of repeating patterns and random behavior. See Figure 4 for an example. There are only 4 out of the 256 basic c.a.'s that exhibit this behavior, and all four are essentially the same when we consider black-white

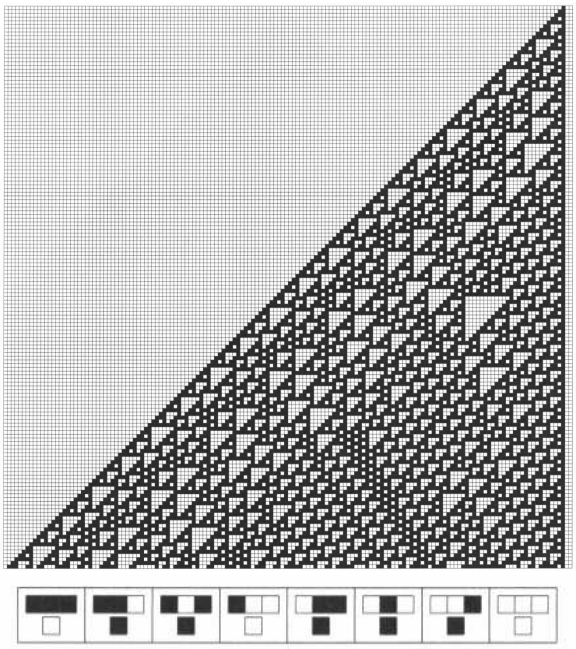


FIGURE 4. Above is an image of 150 iterations of rule 110 and its updating rules. Wolfram, page 32.

symmetry and left-right symmetry. Two of them are mirror images of each other, flipped over the vertical axis placed at the location of the black cell in the initial conditions. The other two are the same as the first two where the state of every cell is reversed, (after the first time step).

The four classifications of behavior are important for the discussion of computational complexity in section 4. Although the classifications are distinguished only by visually recognized patterns and characteristics, it is precisely this that Wolfram believes is main factor in determining levels complexity.

3. Computational Universality

3.1. **The Turing Machine.** Computational universality is the ability of a machine or program to compute the iterations of any other machine or program. It is the concept that gave birth to the computer revolution. Using terminology from modern computation, the universally computational machine is analogous to "hardware", while the possible tasks it can perform (those of other machines) are analogous to "software". Hardware and software differ only in that the first is computationally universal and the second is not.

The concept of computational universality was first discovered by Alan Turing while working with his Turing machines in the 1950's. The Turing machine is meant to perform algorithmic computation by following the steps that a human would employ. The basic steps that one takes are broken down to the essential elements of reading and replacing symbols, moving from symbol to symbol, and the mental states active at each of the previous elements. To replicate the process, a Turing machine uses a set number of symbols, a set number of states, and an infinite tape of cells (the same as one-dimensional c.a.'s) and a "machine head". At each time step, the machine head is over a single cell of the tape. It reads the symbol off the tape, replaces it with another symbol, moves one cell in either direction and changes state.

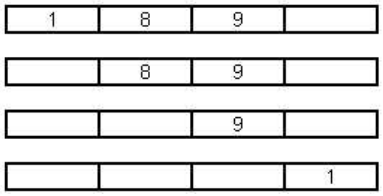
All cells not focused on by the machine head keep the same symbol from step to step. A machine chart exists for each Turing machine that has symbols on one axis and states on the other. When the machine head reads the symbol s while in state g, it will follow the instructions held in square s, g of the machine table. The instructions include the replacement symbol (which may be the existing symbol), the direction to move in and the state to enter. Turing machine tables can also be expressed with images, see Figure 6, (b).

At the beginning of a Turing machine's program, an input of symbols is placed on the tape. By convention, the machine head always begins on the left most input. The head moves back and forth, reading and replacing symbols, following the instructions in the table. The head will either continue indefinitely, or stop when it receives instructions to go into the "final" state. The final state includes no instruction causing the head to freeze.

Figure 5 is a Turing machine that symbolically determines whether a number is odd or even. The instructions are given in the form: replacement symbol, direction (r/l), and new state. Starting on the left most input cell, (the highest place value of the number to be evaluated), the head moves to the right, replacing every symbol with a blank square. Whenever it reads a symbol representative of an odd digit, it goes into

	States			
Symbols	Start	Odd	Even	Final
0	□/r/even	□/r/even	□/r/even	
1	□/r/odd	□/r/odd	□/r/odd	
2	□/r/even	□/r/even	□/r/even	
3	□/r/odd	□/r/odd	□/r/odd	
4	□/r/even	□/r/even	□/r/even	
5	□/r/odd	□/r/odd	□/r/odd	
6	□/r/even	□/r/even	□/r/even	
7	□/r/odd	□/r/odd	□/r/odd	
8	□/r/even	□/r/even	□/r/even	
9	□/r/odd	□/r/odd	□/r/odd	
□	□/right/final	1/right/final	0/right/final	

(a)



(b)

FIGURE 5. Figure (a) is the machine table for a Turing machine that determines whether an input is odd or even. The instructions for the machine head are in the following order: replacement symbol/direction of head movement/new state. Notice that the machine head will replace every symbol with a blank square except in two cases. These cases are when the head encounters a blank square and is in odd or even state. In Figure (b) are four iterations of Turing machine (a) on a three symbol input.

the odd state, and vice a versa for even numbers. When the head comes to a blank square it prints the symbol 1 if it is the odd state, or the symbol 0 if it is in the even state. In both cases it goes into the final state after printing a symbol. The output of the program is a symbolic representation of the numbers status. Machine tables can be created to perform an infinite number of algorithmic computations, some of which coincide with known mathematical algorithms.

It was discovered by Turing that the information in a machine table could be represented in one long strand of symbols. Let this linear representation of the machine table be called $t1$, and let the input that this table would act upon be called $i1$. Turing then discovered that a if $t1$ was placed next to $i1$ on an input tape, then a new machine table,

$t2$, could be written that would perform the instructions of $t1$ on $i1$. This means that the machine head instructed by $t2$ would travel back and forth between $i1$ and $t1$, create the proper output for the input $i1$ and then erase $t1$ from the tape. The machine defined by the table $t2$ is known as the universal Turing machine because it is able to perform the algorithms of any other Turing machine by reading the machine table and input as inputs. The universal Turing machine is even able to perform the algorithms of machines more complicated (more symbols and or more states) than itself. It is a consequence that any algorithm that can be performed by a Turing machine can thus be performed by the universal Turing machine. It is also a consequence that no program can be more computationally complex than a computationally universal machine.

3.2. **Emulation and the Universal Cellular Automata.** Many other programs or machines such as cellular automata, register machines, substitution systems, or tag machines can also be shown to be computationally universal. Because the only existing proof of computational universality pertains to the original Turing machine, all other programs are shown to be computationally universal through emulation. Emulation means that a series of iterations in one program

produces an equivalent representation of every step of the emulated program's computation.

To emulate a Turing machine with a cellular automaton, the iterations of the Turing machine must be displayed vertically as in the c.a.'s discussed above. At each iteration the Turing machine shows the symbols present and the position of the head. In the c.a. that emulates the Turing machine, a color is designated for every possible combination of state and symbol, as well as one color for each symbol when it is not focused on by the head, and thus not connected to a state. Figure 6, (a) shows a Turing machine with two symbols (colors) and three states and the c.a. equivalent. The c.a. that emulates it has eight colors $(2symbols * 3state + 2symbols)$. For organizational purposes, the cells of the Turing machine where the head is not focused, are represented by the two lightest colors in the cellular automaton. The six darker colors represent movement and state of the head. A set of nearest neighbor rules for the computation of the c.a. are then derived from the machine table of the Turing machine. See Figure 6, (b) and (c) for the Turing machine table and the c.a. rules respectively.

This example of emulation is expandable to Turing machines with greater numbers of symbols and states. The number of colors used in the c.a. increase rapidly, as does the number of cases for which rules

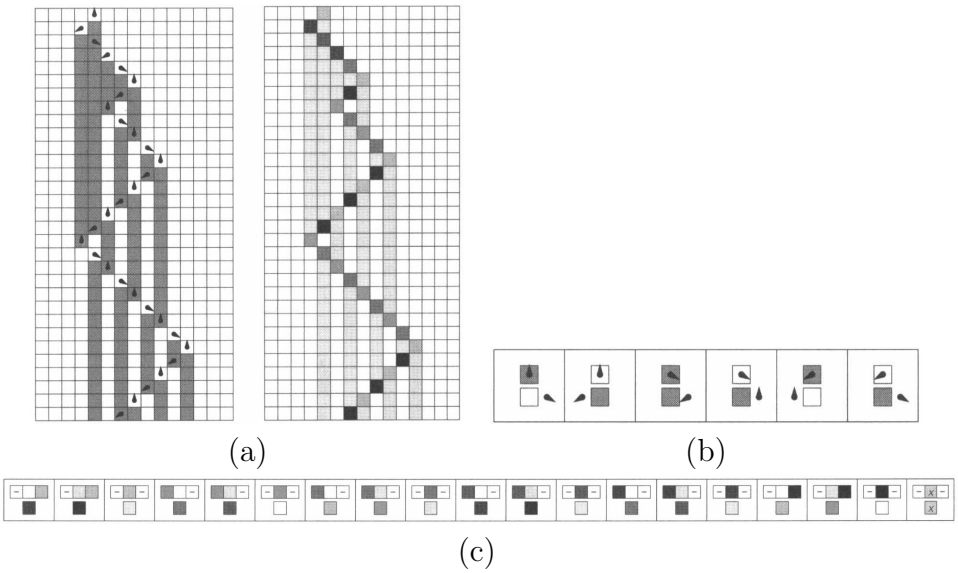


FIGURE 6. Figure (a) is the emulation of a Turing machine by a cellular automaton. Each of the 6 symbol-state combination, as well as both symbol-no-state combinations, are given a color in the c.a. Figure (b) is the machine table for the Turing machine. The pointer and the colors represent the states and the symbols, respectively. Figure (c) shows the c.a. rules derived from the Turing machine table. The white cells with a horizontal line in Figure (c) mean that any color can be in that cell. Wolfram, page 658.

need to be derived. The derivation of c.a. rules remains fairly simple however because only one cell is updated per iteration.

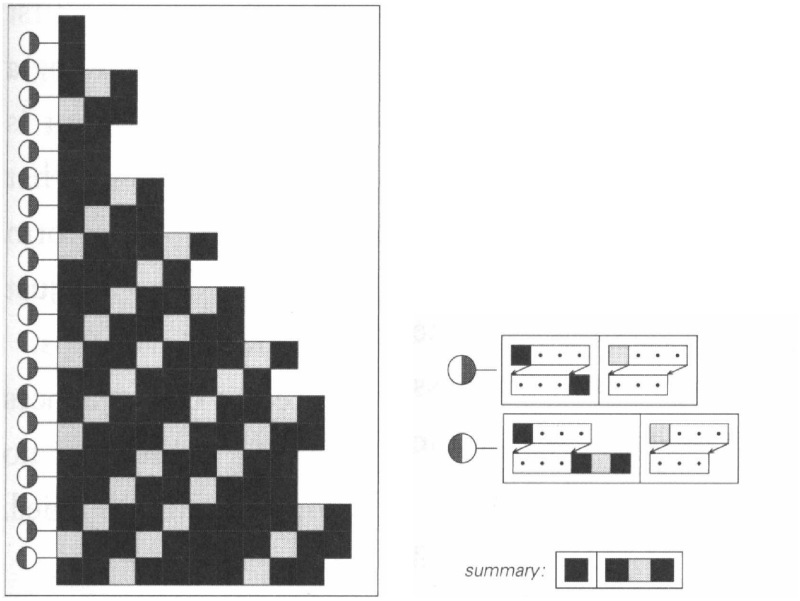
Following this method of emulating Turing machines with cellular automata will lead to extremely complicated c.a.'s for even the simplest of universal Turing machines. Given the capabilities of universally computational machines, as seen in the modern digital computer, this fact does not strike one as particularly odd. The major discovery of

Wolfram's research is that there exist extremely simple programs and machines capable of universal computation. This is demonstrated in the two color, nearest neighbor cellular automaton rule 110 (see Figure 4).

Rule 110 produces Class IV behavior: a combination of randomness and regular repeating structures. Wolfram uses the regular repeating structures to emulate the computation of a class of cyclical tag machines. Some of the cyclical tag machines in this class can emulate universal Turing machines. Rule 110, a member of the most simple cellular automata possible can thus be shown to be universally computational.

A cyclical tag machine is a slightly more complicated version of the standard tag machine. The standard version starts with a single square, either black or white. There are two updating rules, one for when the left most square of the sequence is black, and one for when it is white. At each iteration, the left most square is removed from the sequence and a combination of white and black cells (typically 0-3) is added to the right end of the sequence. A cyclical tag machine is similar, but has two sets of rules that take effect on alternating iterations.

The computation of tag machines are displayed in the same way as in one dimensional c.a.'s and Turing machines, except that the sequence



(a) (b)

FIGURE 7. This is an example of a cyclical tag machine that rule 110 can emulate. In each iteration the symbol in the farthest left column is removed and a sequence is added to the right end of the row. Wolfram, page 679.

is shifted one cell to the left at each iteration so the left most edge lines up.

To show that rule 110 can emulate a cyclical tag machine, Wolfram expands the visual representation to a larger format. The first change is that iterations are shown without shifting at each step. This allows the position of each cell to be maintained by column. The iterations are then separated by a section of blank space.

Wolfram then employs a series of diagonal lines at each iteration to visually represent all the critical information about the machine. This

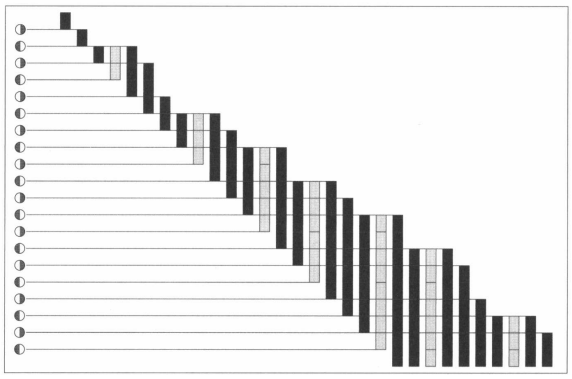


FIGURE 8. The separated iterations where each horizontal line divides iterations. The half colored circle to the left signifies which set of rules is followed at that division. Wolfram, page 679.

information includes which cycle the machine is on, the combination of cells to be added and the method in which the color of the left most cell is determined. The way the lines interact is chosen specifically so that this information is represented visually.

Wolfram then selects a number of regular repeating features from the behavior of rule 110 that interact in similar ways to the information lines of the cyclical tag machine. The process of finding regular repeating objects that perform this objective is done by trial and error and can take a tremendous amount of time. Once it is completed however, one only needs to specify the initial conditions of rule 110 so that these regular repeating shapes are isolated on an otherwise calm plane. The interaction is in general hard to interpret unless seen from a very zoomed out perspective. For the emulation of each horizontal

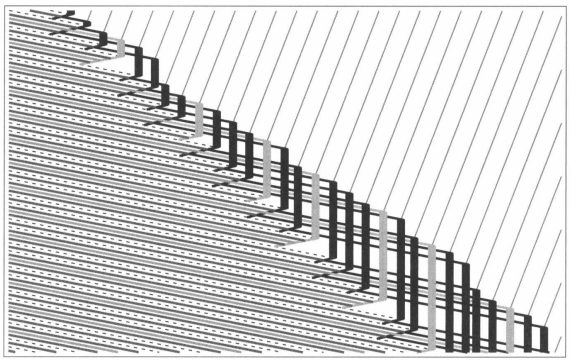


FIGURE 9. The solid lines that come from the left side of the figure represent what the rule will add to the end of the row. In both rule cases, if the solid line(s) hit an extension of a gray first square they stop: nothing is added to the row. When the first square is black, the solid line(s) continue through the extension of the it and add to the end of the row. The lines that come from the upper right portion of the figure interact with the solid lines causing them to stop and create a column. The dotted lines tell the columns when they are the farthest left cell in the row. Wolfram, page 679.

cell in the cyclical tag machine is represented by 3,000 horizontal cells in rule 110. At this range the objects that Wolfram selects can be seen to interact in the same way that the information lines do. Rule 110 is thus able to represent every iteration of the cyclical tag machines computation, effectively an emulation.

This process can be generalized for proving the computational universality of any machine or program. First, one must find a scheme for setting up initial conditions and decoding output so that it can be seen to emulate some other machine known to be universal. The

scheme can be as complicated as necessary without being universal itself (or its use in proving universality is insufficient). Since there are a seemingly infinite number of possible schemes, finding one that can prove computational universality is quite challenging. Proving that a machine or program is not universal, however, is even more challenging because every possible scheme must be checked.

4. The Threshold of Universality

The concept of computational universality implies that once a certain level of complexity is reached, there are no gains in computational ability. This is specifically implied by the ability of the universal Turing machine to emulate behavior of Turing machines with more complicated machine tables than its own. The level of complication at which universality is reached is called the threshold of universality. Traditional intuitions, developed during the computer revolution place this threshold to be quite high. We assume that a machine capable of such complex computation is either made of complicated parts or simple parts put together in a very complex way. The results of section 3 show in fact that one of the 256 simplest cellular automata is universal. The threshold is in fact surprisingly low. Although the cellular automaton rule 110 is the only example in this paper, there are many

other types of machines and programs in *A New Kind of Science* that display universal computation and are just as simple. Almost all of these examples fall into Class IV behavior, while a small handful are of Class III.

The threshold of universality and a number of important similarities between simple programs and systems in nature are the fuel for Wolfram's Principle of Computational Equivalence. The theoretical principle is based around one key idea: "that all processes, whether they are produced by human effort or occur spontaneously in nature, can be viewed as computations." Cellular automata and other such programs are not only capable of the levels of complexity seen in spontaneous natural processes, but also show striking visual similarities. Both natural systems and programs exhibit similarities in behavior while differing significantly in biological base and underlying structure. Wolfram frames these similarities of behavior seen among natural processes, as well as those between programs and natural processes in terms of computation.

The second part of the principle asserts that all systems that do not appear obviously simple, are of equivalent computational sophistication. Essentially, Wolfram theorizes all systems in the universe that exhibit Class III or Class IV behavior are computationally universal

and thus fundamentally equivalent. If all a system does is compute its own behavior, and the system is computationally universal, then it can reproduce any other systems behavior, and all systems are equivalent. Key to this argument is of course that the threshold of universality is reached by all systems producing Class III and Class IV behavior.

Challenges to the principle can come from two angles. The first is that there exist processes more complicated than what is seen in universal programs, such as continuous processes or human thought. Wolfram responds the first example by asserting that it is just as challenging to emulate continuous systems with discrete models as it is to emulate discrete systems with continuous models. This implies that the levels of complexities are similar, if not the same. Wolfram responds to the human thought issue by expressing his belief that advances in neuro science will lead to an understanding of the brain in terms of simple computational programs.

The other challenge is that there exist a number of Class III processes and programs such as rule 30, that are not universal. This is a good argument, but the discussion in 3.2 shows that the process of disproving the presence of universality is harder than proving it. Wolfram speculates that many such Class III rules will be shown to computationally universal in the future.

The implications of the Principle of Computational Equivalence are astounding. Essentially, every process we see around us is a system carrying out a computation, that for the most part (with the exception of the obviously simple systems) is of the same level of computational complexity. The only difference between the computation of cellular automata and natural processes is that we do not know the rules that the natural processes follow. The principle of computational equivalence is so broad in scope, it carries implications for almost every field from evolution, to physics and mathematics, to psychology and philosophy.

5. Modeling With Cellular Automata

5.1. **The Process of Creating a Model.** The Principle of Computational Equivalence has very profound implications for almost every field of scientific and social inquiry. If it is true, one would predict that using c.a.'s or other such programs to model natural processes would be very successful. There is one important concept that makes this issue more complicated, the concept of "computational irreducibility." This means that the behavior of a not-obviously-simple program can not be reduced to a simpler computation (such as a mathematical equation). Essentially, there "exists" no equation that can predict the behavior of

a system at each time step without being as computationally intensive as letting the system compute its own behavior. There are no simpler or lower dimensional representations of these systems. All systems that are complicated enough to break the threshold of universality are computationally irreducible.

Models, however, are simplified representations of natural processes. They condense the information of a process for the purposes of prediction or of understanding behavioral mechanisms. If the principle of computational equivalence is correct then no non-simple process can truly be modeled because of computational irreducibility. No reduction of information is possible.

On the other hand, if all natural processes are computations, we should be able to emulate behavior exactly with another universal system. All one has to do is determine all the information produced by a system at each time step and find an appropriate emulation scheme. Wolfram argues that similarities in the behavior of systems following different governing mechanisms negates the importance of basic elements: whether they are black and white cells, molecules or particle, there is no computational difference.

There exists a middle ground, however, between not being able to make a model and emulating behavior exactly. Although emulation

(through the process mentioned in section 3.2) of natural processes is practically impossible, and reduction of computational information is impossible, there is still modeling potential at the visual level. After viewing the behavior of a large number of simple programs and machines, visual similarities to natural processes such as leaf growth, pigmentation, crystal growth, shell growth are easily identified. The connecting model is created by selecting a specific feature(s) of a process, and adjusting the parameters of c.a. or other program until its behavior matches.

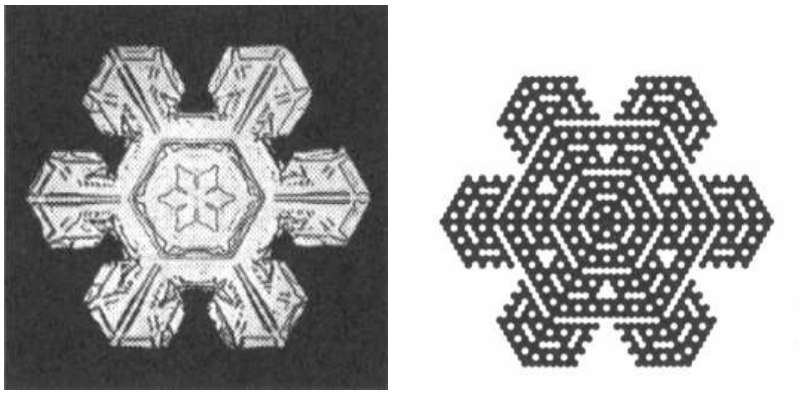
This is the only way to model using c.a.'s because computational irreducibility does not allow for many other forms of analysis. Analysis from traditional mathematics all require some form or reduction of information. Analysis specifically developed to deal with c.a.'s and other such programs is still in the early stages. Even with these drawbacks, the c.a. models in the following section do show some remarkable results.

5.2. **Models.** One such c.a. model developed by Wolfram is that of snowflake growth. The features selected for modeling include the general hexagonal shape and the mechanism of growth that creates the intricate patterns within the hexagon. The mechanism is hypothesized to occur because water particles give off a small bit of heat when they

change from liquid to solid form. When particles add to the crystal during growth, heat is released prohibiting future growth in the area for a time. This process results in the gaps in between parts of the snowflake.

The c.a. model of this process is set on an infinite two dimensional hexagonal plane to ensure the hexagonal feature of the overall shape. The cells are either black or white and only adjacent cells are considered to be in a cells neighborhood. The updating rules (in order to incorporate the process of heat release mentioned above), allow a white cell to turn black if only one cell in its neighborhood is black. If there are more than one black cells in the neighborhood, the white cell remains white. Figure 10 shows both an actual snowflake and iteration 24 of the c.a. model. Although the images are not identical, the selected visual features have been successfully captured.

Another model developed in *A New Kind of Science* is that of fluid as it interacts with a barrier at various speeds. The features of fluid flow being modeled are the various patterns of eddy formation as otherwise undisturbed fluid encounters a solid object. At low speeds the fluid slides around the obstruction with little alteration. As speed increases, the fluid directly behind the obstruction moves slower than the rest creating a pair of eddies behind object. As the speed of the fluid



(a) (b)

FIGURE 10. A picture of a snow flake and the 24^{th} iteration of a two color hexagonal c.a. The updating rules of the c.a. allow a white cell to turn black if only one cell in its neighborhood is black. If there are more than one black cells in the neighborhood, the white cell remains white. Wolfram, pages 370-371.

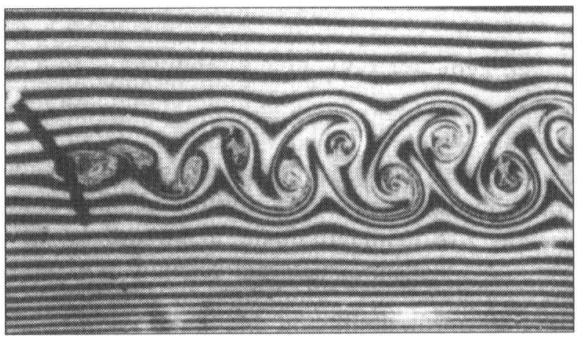


FIGURE 11. This is a picture of a vortex street disturbed by an obstacle. Wolfram, page 377.

continues to increase, the eddies begin to travel with the fluid. New eddies occur in the original locations and a trail of connecting eddies appears behind the object, see Figure 11.

The c.a. model of this process is also set on an infinite plane of hexagonal cells. Each hexagon is broken into six equilateral triangles. The

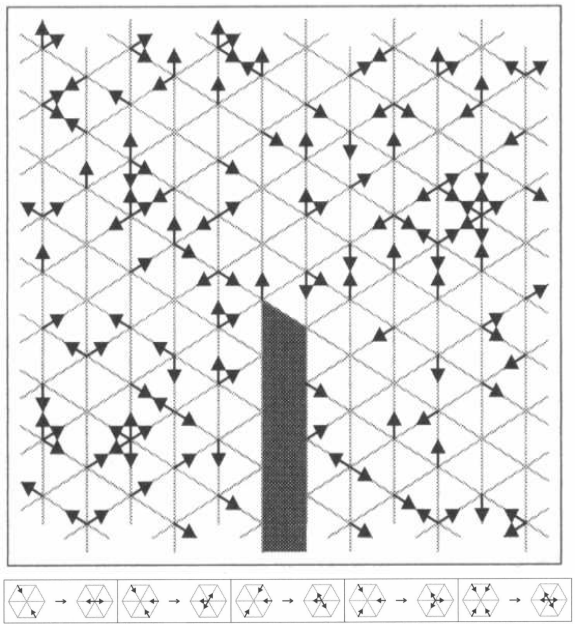


FIGURE 12. This is a picture of the liquid dynamic model at the most basic level. The fluid flow is from left to right. Also shown are the updating rules. Wolfram, page 378.

lattice of cells in this c.a. serve as a frame work for thousands of small vectors that represent water molecules. See Figure 12 for the updating rules and an iteration including the top end of the obstruction. The rules determine how vectors will leave a vertex based on their incoming direction and quantity. In the model, new vectors are continually added to the plane from the left. The frequency at which they are added represents the speed of the fluid being modeled.

The density of the vectors is $1/6^{th}$ of the maximum which Wolfram equates to a Reynolds number of 100. The results are strikingly similar to fluid flow of this density.

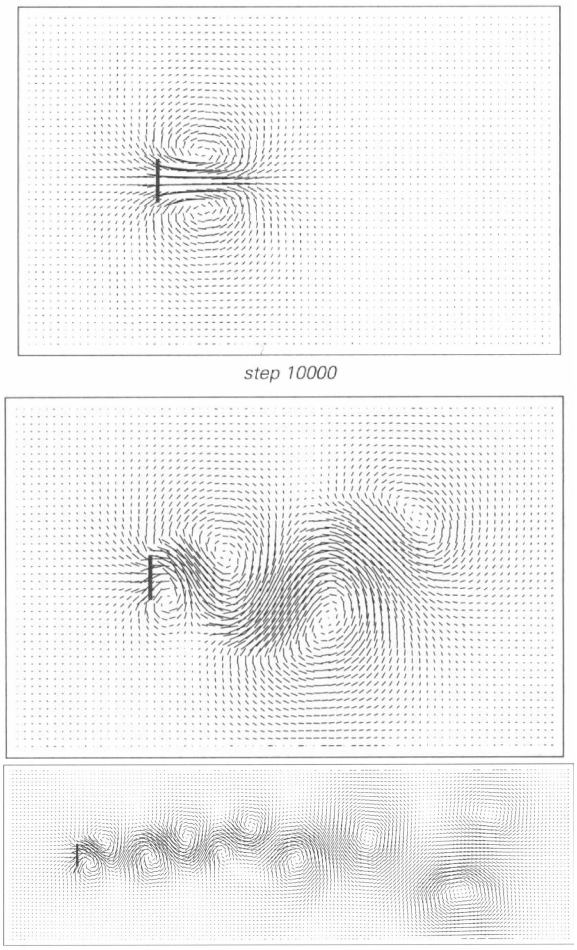


FIGURE 13. The three images above are all the same c.a., but at iterations 1,000, 4,000, and 7,000, from top to bottom. Each line vector is an average velocity vector of a 20X20 cell black. The vectors enter from the left in a regular way with a frequency that represents 0.4 of maximum speed. Wolfram, page 380.

Both of these models are pretty successful in capturing the selected features of the natural processes. Many pressing questions remain however. Such as: How do I compare the accuracy of this model to another of the same process? Does this model have any predictive capabilities?

and, What does this model actually tell us about the natural process? These questions have no answers because of the limited forms of analysis available for programs of this type.

6. Conclusion

Stephen Wolfram's research on the behavior of simple programs and the concept of computational universality have great significance for the field of computer science. Traditional assumptions about the structural complexity required of a computationally universal system have been proved false, warranting continued study of the phenomenon. Besides adding to the annals of computer science, the results of Wolfram's research also shows potential for computer based technology.

The Principle of Computational Equivalence is where the conclusions about the nature of computation and complexity are applied to systems other than those existing inside of a computer. All that is learned about computational processes and the threshold of universality is hypothesized to be present in every natural system in the universe. The principle is both incredibly broad and profound, specifically when one carries out the logical implications held in the principle for philosophy, physics, evolutionary biology and psychology. The impetus for

the principle comes from the striking similarities between various natural processes and behavior seen in computational systems. There is no question that the modeling of natural processes with computational systems supports the principle. The evidence is not sufficient for proof however. The insufficiency of the evidence is a product of the shortage of analytical tools available for studying computationally irreducible programs. There is currently no way to concretely relate these types of programs to traditional mathematics.

In the future, we will find the Principle of Computational Equivalence to be either correct, incorrect, or logically beyond proof. The result will depend on the issue of analysis discussed above. If no form of analysis arises that can create a more concrete connection between natural systems, computational systems and traditional mathematics, the principle will remain unprovable and understood only in its own terminology.

Regardless of the outcome,*A New Kind of Science* is a landmark study of the nature of complexity and randomness.

References

[1] Wolfram, Stephen. A New Kind of Science. Champaign, IL: Wolfram Media Inc., 2002.

[2] Davis, Martin. The Universal Computer: The Road from Leibniz to Turing. New York: Norton, 2000.