EXCERPTED FROM

STEPHEN WOLFRAM

A NEW KIND OF SCIENCE

SECTION 12.4

The Validity of the Principle

In a pattern like the one obtained from rule 30 above different computations are presumably not arranged in any such straightforward way. But I strongly suspect that even though it may be quite impractical to find particular computations that one wants, it is still the case that essentially any possible computation exists somewhere in the pattern.

Much as in the case of universality for complete systems, however, the Principle of Computational Equivalence does not just say that a sophisticated computation will be found somewhere in a pattern produced by a system like rule 30. Rather, it asserts that unless it is obviously simple essentially any behavior that one sees should correspond to a computation of equivalent sophistication.

And in a sense this can be viewed as providing a new way to define the very notion of computation. For it implies that essentially any piece of complex behavior that we see corresponds to a kind of lump of computation that is at some level equivalent.

It is a little like what happens in thermodynamics, where all sorts of complicated microscopic motions are identified as corresponding in some uniform way to a notion of heat.

But computation is both a much more general and much more powerful notion than heat. And as a result, the Principle of Computational Equivalence has vastly richer implications than the laws of thermodynamics—or for that matter, than essentially any single collection of laws in science.

The Validity of the Principle

With the intuition of traditional science the Principle of Computational Equivalence—and particularly many of its implications—might seem almost absurd. But as I have developed more and more new intuition from the discoveries in this book so I have become more and more certain that the Principle of Computational Equivalence must be valid.

But like any principle in science with real content it could in the future always be found that at least some aspect of the Principle of Computational Equivalence is not valid. For as a law of nature the principle could turn out to disagree with what is observed in our

universe, while as an abstract fact it could simply represent an incorrect deduction, and even as a definition it could prove not useful or relevant.

But as more and more evidence is accumulated for phenomena that would follow from the principle, so it becomes more and more reasonable to expect that at least in some formulation or another the principle itself must be valid.

As with many fundamental principles the most general statement of the Principle of Computational Equivalence may at first seem quite vague. But almost any specific application of the principle will tend to suggest more specific and precise statements.

Needless to say, it will always be possible to come up with statements that might seem related to the Principle of Computational Equivalence but are not in fact the same. And indeed I suspect this will happen many times over the years to come. For if one tries to use methods from traditional science and mathematics it is almost inevitable that one will be led to statements that are rather different from the actual Principle of Computational Equivalence.

Indeed, my guess is that there is basically no way to formulate an accurate statement of the principle except by using methods from the kind of science introduced in this book. And what this means is that almost any statement that can, for example, readily be investigated by the traditional methods of mathematical proof will tend to be largely irrelevant to the true Principle of Computational Equivalence.

In the course of this book I have made a variety of discoveries that can be interpreted as limited versions of the Principle of Computational Equivalence. And as the years and decades go by, it is my expectation that many more such discoveries will be made. And as these discoveries are absorbed, I suspect that general intuition in science will gradually shift, until in the end the Principle of Computational Equivalence will come to seem almost obvious.

But as of now the principle is far from obvious to most of those whose intuition is derived from traditional science. And as a result all sorts of objections to the principle will no doubt be raised. Some of them will presumably be based on believing that actual systems have

less computational sophistication than is implied by the principle, while others will be based on believing that they have more.

But at an underlying level I suspect that the single most common cause of objections will be confusion about various idealizations that are made in traditional models for systems. For even though a system itself may follow the Principle of Computational Equivalence, there is no guarantee that this will also be true of idealizations of the system.

As I discussed at the beginning of Chapter 8, finding a good model for a system is mostly about finding idealizations that are as simple as possible, but that nevertheless still capture the important features of the system. And the point is that in the past there was never a clear idea that computational capabilities of systems might be important, so these were usually not captured correctly when models were made.

Yet one of the characteristics of the kinds of models based on simple programs that I have developed in this book is that they do appear successfully to capture the computational capabilities of a wide range of systems in nature and elsewhere. And in the context of such models what I have discovered is that there is indeed all sorts of evidence for the Principle of Computational Equivalence.

But if one uses the kinds of traditional mathematical models that have in the past been common, things can seem rather different.

For example, many such models idealize systems to the point where their complete behavior can be described just by some simple mathematical formula that relates a few overall numerical quantities. And if one thinks only about this idealization one almost inevitably concludes that the system has very little computational sophistication.

It is also common for traditional mathematical models to suggest too much computational sophistication. For example, as I discussed at the end of Chapter 7, models based on traditional mathematical equations often give constraints on behavior rather than explicit rules for generating behavior.

And if one assumes that actual systems somehow always manage to find ways to satisfy such constraints, one will be led to conclude that these systems must be computationally more sophisticated than any of

the universal systems I have discussed—and must thus violate the Principle of Computational Equivalence.

For as I will describe in more detail later in this chapter, an ordinary universal system cannot in any finite number of steps guarantee to be able to tell whether, say, there is any pattern of black and white squares that satisfies some constraint of the type I discussed at the end of Chapter 5. Yet traditional mathematical models often in effect imply that systems in nature can do things like this.

But I explained at the end of Chapter 7 this is presumably just an idealization. For while in simple cases complicated molecules may for example arrange themselves in configurations that minimize energy, the evidence is that in more complicated cases they typically do not. And in fact, what they actually seem to do is instead to explore different configurations by an explicit process of evolution that is quite consistent with the Principle of Computational Equivalence.

One of the features of cellular automata and most of the other computational systems that I have discussed in this book is that they are in some fundamental sense discrete. Yet traditional mathematical models almost always involve continuous quantities. And this has in the past often been taken to imply that systems in nature are able to do computations that are somehow fundamentally more sophisticated than standard computational systems.

But for several reasons I do not believe this conclusion.

For a start, the experience has been that if one actually tries to build analog computers that make use of continuous physical processes they usually end up being less powerful than ordinary digital computers, rather than more so.

And indeed, as I have discussed several times in this book, it is in many cases clear that the whole notion of continuity is just an idealization—although one that happens to be almost required if one wants to make use of traditional mathematical methods.

Fluids provide one obvious example. For usually they are thought of as being described by continuous mathematical equations. But at an underlying level real fluids consist of discrete particles. And this means that whatever the mathematical equations may suggest, the actual

ultimate computational capabilities of fluids must be those of a system of discrete particles.

But while it is known that many systems in nature are made up of discrete elements, it is still almost universally believed that there are some things that are fundamentally continuous—notably positions in space and values of quantum mechanical probability amplitudes.

Yet as I discussed in Chapter 9 my strong suspicion is that at a fundamental level absolutely every aspect of our universe will in the end turn out to be discrete. And if this is so, then it immediately implies that there cannot ever ultimately be any form of continuity in our universe that violates the Principle of Computational Equivalence.

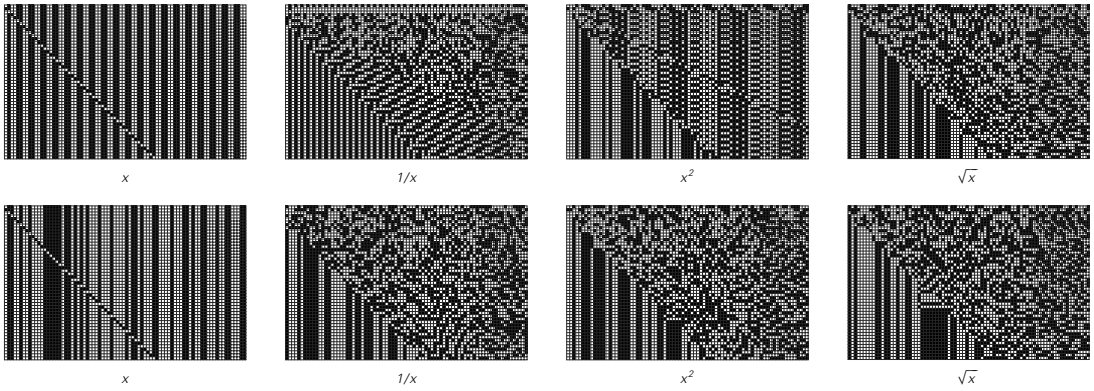
But what if one somehow restricts oneself to a domain where some particular system seems continuous? Can one even at this level perform more sophisticated computations than in a discrete system?

My guess is that for all practical purposes one cannot. Indeed, it is my suspicion that with almost any reasonable set of assumptions even idealized perfectly continuous systems will never in fact be able to perform fundamentally more sophisticated computations.

In a sense the most basic defining characteristic of continuous systems is that they operate on arbitrary continuous numbers. But just to represent every such number in general requires something like an infinite sequence of digits. And so this implies that continuous systems must always in effect be able to operate on infinite sequences.

But in itself this is not particularly remarkable. For even a one-dimensional cellular automaton can be viewed as updating an infinite sequence of cells at every step in its evolution. But one feature of this process is that it is fundamentally local: each cell behaves in a way that is determined purely by cells in a local neighborhood around it.

Yet even the most basic arithmetic operations on continuous numbers typically involve significant non-locality. Thus, for example, when one adds two numbers together there can be carries in the digit sequence that propagate arbitrarily far. And if one computes even a function like $1/x$ almost any digit in x will typically have an effect on almost any digit in the result, as the pictures on the facing page indicate.



Results from mathematical operations on numbers with similar digit sequences. Each successive line in each picture gives the digit sequence obtained by using a value of x in which one successive digit has been reversed. The top row of pictures start from the repetitive base 2 digit sequence of $x = 3/5$; the bottom row of pictures from $x = \pi/4$. The lack of coherence between successive digit sequences in each picture reflects the non-locality of mathematical operations when applied to digit sequences.

But can this detailed kind of phenomenon really be used as the basis for doing fundamentally more sophisticated computations? To compare the general computational capabilities of continuous and discrete systems one needs to find some basic scheme for constructing inputs and decoding outputs that one can use in both types of systems. And the most obvious and practical approach is to require that this always be done by finite discrete processes.

But at least in this case it seems fairly clear that none of the simple functions shown above can for example ever lead to results that go beyond ones that could readily be generated by the evolution of ordinary discrete systems. And the same is presumably true if one works with essentially any of what are normally considered standard mathematical functions. But what happens if one assumes that one can set up a system that not only finds values of such functions but also finds solutions to arbitrary equations involving them?

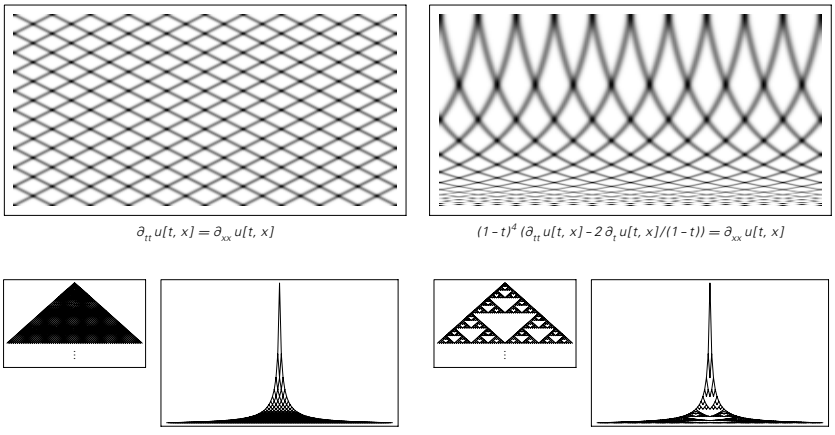
With pure polynomial equations one can deduce from results in algebra that no fundamentally more sophisticated computations become possible. But as soon as one even allows trigonometric functions, for example, it turns out that it becomes possible to construct equations for which finding a solution is equivalent to finding

the outcome of an infinite number of steps in the evolution of a system like a cellular automaton.

And while these particular types of equations have never seriously been proposed as idealizations of actual processes in nature or elsewhere, it turns out that a related phenomenon can presumably occur in differential equations—which represent the most common basis for mathematical models in most areas of traditional science.

Differential equations of the kind we discussed at the end of Chapter 4 work at some level a little like cellular automata. For given the state of a system, they provide rules for determining its state at subsequent times. But whereas cellular automata always evolve only in discrete steps, differential equations instead go through a continuous process of evolution in which time appears just as a parameter.

And by making simple algebraic changes to the way that time enters a differential equation one can often arrange, as in the pictures below, that processes that would normally take an infinite time will actually always occur over only a finite time.



Indications of how an infinite amount of computational work can in principle be performed in a finite time in continuous systems like partial differential equations. The top left picture shows a solution to the wave equation. The top right picture shows a solution to an equation obtained from the wave equation by transforming the time variable according to $t \to 1 - 1/t$. The bottom row shows what the same transformation does to patterns of the kind that are generated by simple cellular automata. It is presumably possible to construct partial differential equations that give both the original and transformed versions of these patterns.

So if such processes can correspond to the evolution of systems like cellular automata, then it follows at least formally that differential equations should be able to do in finite time computations that would take a discrete system like a cellular automaton an infinite time to do.

But just as it is difficult to make an analog computer faithfully reproduce many steps in a discrete computation, so also it seems likely that it will be difficult to set up differential equations that for arbitrarily long times successfully manage to emulate the precise behavior of systems like cellular automata. And in fact my suspicion is that to make this work will require taking limits that are rather similar to following the evolution of the differential equations for an infinite time.

So my guess is that even within the formalism of traditional continuous mathematics realistic idealizations of actual processes will never ultimately be able to perform computations that are more sophisticated than the Principle of Computational Equivalence implies.

But what about the process of human thinking? Does it also follow the Principle of Computational Equivalence? Or does it somehow manage to do computations that are more sophisticated than the Principle of Computational Equivalence implies?

There is a great tendency for us to assume that there must be something extremely sophisticated about human thinking. And certainly the fact that present-day computer systems do not emulate even some of its most obvious features might seem to support this view. But as I discussed in Chapter 10, particularly following the discoveries in this book, it is my strong belief that the basic mechanisms of human thinking will in the end turn out to correspond to rather simple computational processes.

So what all of this suggests is that systems in nature do not perform computations that are more sophisticated than the Principle of Computational Equivalence allows. But on its own this is not enough to establish the complete Principle of Computational Equivalence. For the principle also implies a lower limit on computational sophistication—making the assertion that almost any process that is not obviously simple will tend to be equivalent in its computational sophistication.

And one of the consequences of this is that it implies that most systems whose behavior seems complex should be universal. Yet as of now we only know for certain about fairly few systems that are universal, albeit including ones like rule 110 that have remarkably simple rules. And no doubt the objection will be raised that other systems whose behavior seems complex may not in fact be universal.

In particular, it might be thought that the behavior of systems like rule 30—while obviously at least somewhat computationally sophisticated—might somehow be too random to be harnessed to allow complete universality. And although in Chapter 11 I did give a few pieces of evidence that point towards rule 30 being universal, there can still be doubts until this has been proved for certain.

And in fact there is a particularly abstruse result in mathematical logic that might be thought to show that systems can exist that exhibit some features of arbitrarily sophisticated computation, but which are nevertheless not universal. For in the late 1950s a whole hierarchy of systems with so-called intermediate degrees were constructed with the property that questions about the ultimate output from their evolution could not in general be answered by finite computation, but for which the actual form of this output was not flexible enough to be able to emulate a full range of other systems, and thus support universality.

But when one examines the known examples of such systems—all of which have very intricate underlying rules—one finds that even though the particular part of their behavior that is identified as output is sufficiently restricted to avoid universality, almost every other part of their behavior nevertheless does exhibit universality—just as one would expect from the Principle of Computational Equivalence.

So why else might systems like rule 30 fail to be universal? We know from Chapter 11 that systems whose behavior is purely repetitive or purely nested cannot be universal. And so we might wonder whether perhaps some other form of regularity could be present that would prevent systems like rule 30 from being universal.

When we look at the patterns produced by such systems they certainly do not seem to have any great regularity; indeed in most

respects they seem far more random than patterns produced by systems like rule 110 that we already know are universal.

But how can we be sure that we are not being misled by limitations in our powers of perception and analysis—and that an extraterrestrial intelligence, for example, might not immediately recognize regularity that would show that universality is impossible?

For as we saw in Chapter 10 the methods of perception and analysis that we normally use cannot detect any form of regularity much beyond repetition or at most nesting. So this means that even if some higher form of regularity is in fact present, we as humans might never be able to tell.

In the history of science and mathematics both repetition and nesting feature prominently. And if there was some common higher form of regularity its discovery would no doubt lead to all sorts of important new advances in science and mathematics.

And when I first started looking at systems like cellular automata I in effect implicitly assumed that some such form of regularity must exist. For I was quite certain that even though I saw behavior that seemed to me complex the simplicity of the underlying rules must somehow ultimately lead to great regularity in it.

But as the years have gone by—and as I have investigated more and more systems and tried more and more methods of analysis—I have gradually come to the conclusion that there is no hidden regularity in any large class of systems, and that instead what the Principle of Computational Equivalence suggests is correct: that beyond systems with obvious regularities like repetition and nesting most systems are universal, and are equivalent in their computational sophistication.

Explaining the Phenomenon of Complexity

Early in this book I described the remarkable discovery that even systems with extremely simple underlying rules can produce behavior that seems to us immensely complex. And in the course of this book, I have shown a great many examples of this phenomenon, and have