

Can't Decide? Undecide!

Chaim Goodman-Strauss

In my mathematical youth, when I first learned of Gödel's Theorem, and computational undecidability, I was at once fascinated and strangely reassured of our limited place in the grand universe: incredibly mathematics itself establishes limits on mathematical knowledge. At the same time, as one digs into the formalisms, this area can seem remote from most areas of mathematics and irrelevant to the efforts of most work-a-day mathematicians. But that's just not so! Undecidable problems surround us, everywhere, even in recreational mathematics!

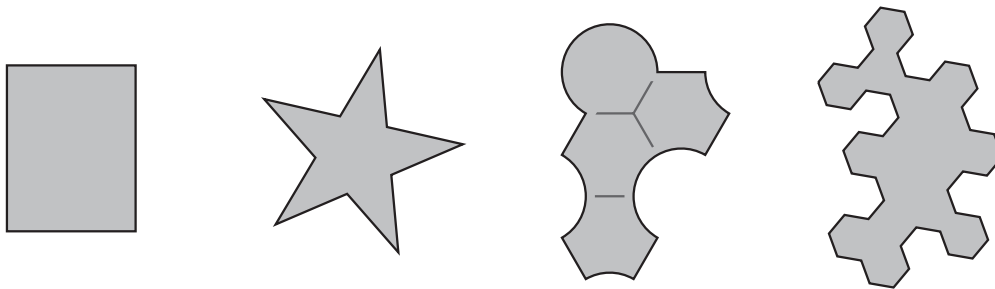
1 Three Mysterious Examples

Somehow these simple questions seem difficult to resolve:

1.1 Mysterious Example #1

Tilings are a rich source of combinatorial puzzles. We can ask, for a given tile, whether or not it *admits a tiling*: that is, does there exist a tiling of the plane by copies of this tile?

For many examples, this is utterly trivial: clearly the tile at left in the figure below does admit a tiling, and the tile at middle left does not. One might discover a simple proof that the tile at middle right does not admit a tiling either¹, though it is more difficult to work out just how large a region you can cover before getting stuck. But it's a reasonable bet that you will not be able to discover whether or not the tile at right, discovered by J. Myers in 2003, admits a tiling, at least not without resorting to some sort of brute-force calculation on a computer! Try this for yourself! A downloadable file with tiles to cut out and play with has been placed at http://mathfactor.uark.edu/downloads/myers_tile.pdf



In general, then, we have

Input: A tile.

and a

Decision Problem: *Does the given tile admit a tiling of the plane?*

¹Hint: the tile, discovered by C. Mann, can be viewed as a cluster of hexagons, with some edges bulging inwards and some bulging out — but there are more bulging inwards than outwards. Etc...

With enough brute-force effort, in some circumstances, we can answer this problem:

We might simply enumerate all possible configurations admitted by the tile, covering larger and larger disks. If the tile *does not* admit a tiling, eventually there will be some sized disk we can no longer cover, we run out of configurations to enumerate, and we then know the answer to our problem: **No**, the tile fails to admit a tiling. If a tile does not admit a tiling, the “Heesch number” is a measure of the complexity of such a tile, as the largest possible combinatorial radius of disks it can cover (in other words, the maximum number of concentric rings copies of the tile can form); C. Mann discovered the current world record examples, with Heesch number 5 [22, 23]. But how can we determine whether an arbitrary given tile *does* admit a tiling?

We can modify our procedure just a bit to discover if a tile admits a *periodic* tiling:² As we enumerate larger and larger configurations, we check to see if we have yet come across one that can serve as a fundamental domain in a periodic tiling. If we find such a configuration we have the answer: **Yes**, the tile admits a tiling. The “isohedral number” is a measure of the complexity of this, as the minimum number of tiles required to form a fundamental domain (that is, the minimum number of orbits in a tiling by such a tile). J. Myers has found many bizarre examples, including the world record example, with isohedral number 10, shown in the figure above at right [30].

If it were true that every tile either admits a periodic tiling, or does not admit a tiling at all, then we would have a procedure to settle our decision problem for any given tile — enumerate larger and larger configurations until we run out or find a fundamental domain. But could there exist an “aperiodic” tile, one admitting only non-periodic tilings?

Almost unimaginably, could it be that there is no possible systematic method to answer our decision problem? Honestly — how hard do you suspect this could be?

If that problem were in fact *undecidable*, then we would have immediate, difficult-to-believe corollaries: There must exist an aperiodic tile — a tile which somehow wrecks translational symmetry at all scales. There cannot be a bound on Heesch number — for any N there must be a tile that can form at least N concentric rings, but then somehow get stuck and never can be continued to form a tiling.

Experimenting with some of the stranger examples discovered by Mann and Myers might give one pause — it seems utterly baffling to discern how and why these examples behave as they do, and others don’t.

1.2 Mysterious Example #2

A large literature on the Collatz function (cf. [19, 21, 25]) has not settled this seemingly simple problem:

Input: *A counting number n .*

Repeatedly we apply the following function to our current n , obtaining a new number n at each step: if n is odd, take $3n + 1$; if our n is even take $n/2$; we halt if we ever obtain $n = 1$.

For example, if we begin with, say $n = 7$, we obtain 22, then 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2 and finally 1.³

Decision Problem: *On a given input, do we obtain 1 and halt, or do we might enter into a loop, or do we run forever, obtaining larger and larger numbers in the long run?*

For a taste of how vexing the behavior of this process can be, work out by hand what occurs beginning with $n = 27$. Again we see our numbers grow and shrink, seemingly without rhyme or reason, until, finally, thankfully!, the process terminates after 111 steps, at one point reaching $n = 9232$.

As in the previous example, we can eventually determine, with enough patience, whether the process halts or loops on a particular given input. As of this writing, this process is known to halt on every input less than $5 \cdot 2^{60}$ [32]! Yet there is no obvious means to determine whether the process runs forever.

²That is, a tiling invariant under some translation. In the Euclidean plane — though not in higher dimensions nor in the hyperbolic plane — if a tile admits such a tiling, it must in fact admit a tiling with a compact fundamental domain.[16]

³Of course if we allowed the process to run forever we would then have 4, 2 and again 1, *ad infinitum*.

If we could prove, once and for all, that this process always halts, the decision problem is instantly solved for all input. But could it be that there is no theorem possible that declares, definitively, that this process always halts? Paul Erdős is said to have remarked “Mathematics is not yet ready for such confusing, troubling, and hard problems.”

1.3 Mysterious Example #3

The logician Emil Post explored this system as a student in the early 1920's [33]:

Input: *A string of 0's and 1's.*

We will repeatedly cross off three digits from the front of our string, and tack on digits at the end, by the following rule: If the first crossed-off digit is 0, we tack on 00; if the first crossed-off digit is 1, we tack on 1101.

There are three possibilities: either our string at some point will have too few letters to cross off, and we *halt*; or we might enter into a *loop*, in which the same strings recur again and again *ad infinitum*; or we might run forever, without looping, the strings eventually growing without bound.

For example, if we begin with the string 10101, we cross off 101 from the left and tack on 1101 on the right, obtaining ~~101~~01 1101 = 011101. Repeating this process we obtain ~~011~~101 00 = 10100, then 001101 and then once again 10100. So on our third string we enter into a loop of length 2.

Decision Problem: *For a given input, does this process loop, halt or run forever?*

Running through a few examples, we quickly see how vexing this question is! As a nice exercise, consider 1·1· (where each · can be either 0 or 1 — since those will be crossed off without rising to the front of the string, it cannot matter either way).

Rummaging through other small examples by hand, one is hard-pressed to find a pattern, and soon one reaches the limits of one's patience:

On input 1·1·1·0·, we run for a whopping 419 steps, before finally reaching 00 and halting.

On input 1·1·1·0·1·0·, we run for 2137 steps before entering into a loop of length 28. Along the way, the strings grow and shrink in a most confounding manner.

Worse yet, seemingly similar inputs, such as 1·1·0·1·0·1· (which halts after just 32 steps) give rise to much simpler behavior. Can you explain *why*?

Certainly, with enough patience, we can determine whether the process will halt or loop on a given input string — simply run the process until one of these two events occurs. But there is no obvious way to determine if we do *not* halt or loop! Is there a procedure that can settle this question for any given input, in a finite number of steps? Post dryly remarks the problem has “proven intractible.” [33]

2 Undecidability

No one knows how to answer the decision problems above. In each case, we can answer the problem for some inputs, and seem to be stuck on others. In each case, the behavior of the problem, on each given input, can be rather unexpected, and small changes to the input can cause our process to play out in radically different ways.

In each case, we might throw up our hands and ask whether a general technique for answering the decision problem is even possible, whether one might find a theorem that could classify, in some effective manner, for which inputs the problem is answered one way, and for which, another. No one even knows, though, whether the mysterious examples above are in fact *undecidable*.

Quite remarkably, as many readers will of course know, there are in fact decision problems for which one

can *prove* no mechanical process — or procedure, or algorithm — can provide an answer on any given input, problems one can *prove* are undecidable. ⁴

I'll go out on a limb and state my belief that one of our mysterious examples is almost certainly undecidable, one seems not likely to be, and one I have no idea. But of course I would be quite rash if I said which is which.

Our main point here — which seems to be less widely appreciated — is that *undecidable problems are in a sense ubiquitous*, arising even in elementary, recreational settings.

There are hundreds of interesting and useful treatments of this subject: an excellent beginning is Sipser's *Introduction to the Theory of Computation* [39]. Minsky's classic *Computation: Finite and Infinite Machines* [29] provides valuable context and constructions. Together with its encyclopedic endnotes, Wolfram's *A New Kind of Science* [45] is a definitive sourcebook of specific, simple examples. The *Wikipedia* entries in this area are comprehensive and generally well-written and accurate. Margenstern's helpful survey reviews the recent state-of-the-art of our knowledge of the frontier between decidability and undecidability [24]. Smullyan's many puzzle books, particularly *The Lady and the Tiger* [40], pose these issues in fun ways that can excite very young mathematicians. Finally, Hofstadter's Pulitzer Prize winning, delightful *Gödel, Escher, Bach* [18] continues to earn a wider audience for this subject.

The three mysterious examples each generalize to problems known to be undecidable:

2.1 Undecidable Tiling Problems

Undecidability has a long pedigree in recreational mathematics. In 1961 as an aside within his work on one of the then remaining open cases of Hilbert's *Entscheidungsproblem* ("Is a given first order logical formula satisfiable?") [4, 44], Hao Wang noted the undecidability of a particular elementary tiling problem. This began a course of development that led straight to the discovery of various "aperiodic" sets of tiles, most famously Penrose's, popularized by Martin Gardner [14].

Input: *A finite collection of tiles, and a particular "seed" configuration.*

Decision Problem: *Do the given tiles admit a tiling of the plane which contains the given configuration?*

That is, can we "complete" the seed configuration to form a tiling of the entire plane, using copies of some or all of the given tiles?

Wang accomplished this by reducing the "Halting Problem" for Turing machines to his Completion Problem: For any given Turing machine, he produced a set of tiles and seed configuration, in such a way that the seed configuration could be extended to a tiling by copies of the tiles if and only if the machine never halts.

Of course the Halting Problem is a touchstone of undecidability: in 1935, through a simple and elegant construction, Turing proved there can be no procedure to decide whether a given Turing machine will halt or not [43]; consequently, there can be no procedure to tell whether one of Wang's corresponding sets of tiles, with its seed configuration, can complete a tiling of the plane.

Wang's construction is easy enough to illustrate by an example: Consider the Turing machine specified by

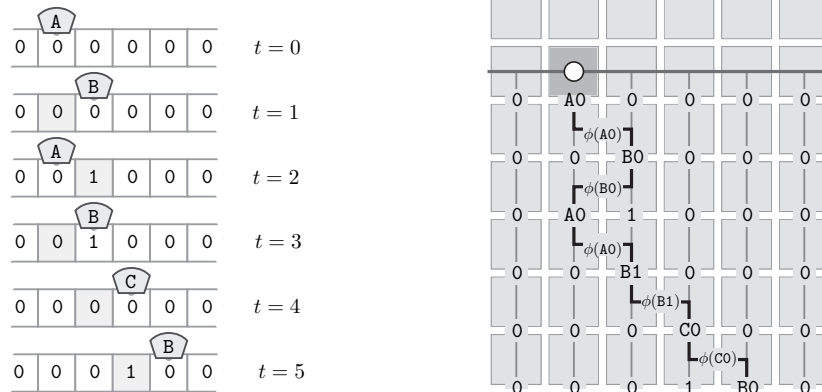
ϕ	A	B	C
0	ORB	1LA	1RB
1	1RB	ORC	OLH

This machine will work on an infinite tape; at each step, each cell is marked 0 or 1 and the machine will be in a particular state A, B or C, reading one particular cell. The transition function ϕ determines the action

⁴This should not be confused with the use of the word "undecidable" to mean that a given statement is independent of a specific formal deductive system, as for example the Continuum Hypothesis is independent of Zermelo-Fraenkel set theory.

of the machine, depending on its state and the marking it is reading; for example, if the machine is in state A reading 0, as in the upper left of the table, the machine will leave a 0 in that spot on the tape, move right one cell, and go into state B. If it is in state B reading a 0, it leaves 1 on the tape, moves one cell to the left, and goes into state A. There is one special “halt” state H— if the machine enters this state, it can do no more, and the process halts.

Beginning in state A, on a tape marked with all 0’s, we can illustrate the first few steps of the run of the machine, at left below.

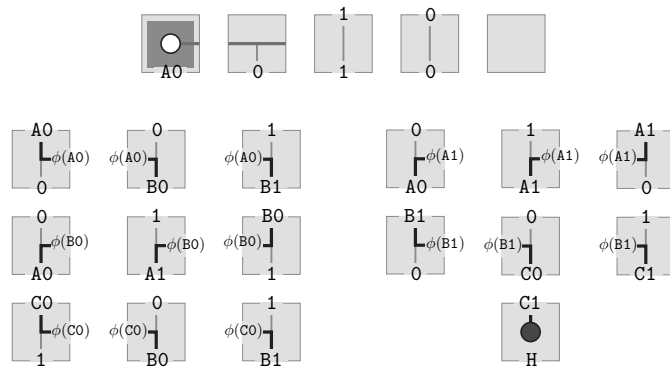


The essential point is that this illustration itself satisfies completely local rules: it is composed of pieces that must fit together in a certain manner. We can encode this as a tiling, shown at right above.


With only a little care, we then have a set of tiles, shown below, that *can* emulate the machine. It is possible to cover the plane with copies of these tiles, so that labels on adjacent edges match⁵, as shown above.

Must they emulate this machine? In any tiling containing the initial “seed tile”, at upper left below, there must be, inductively row by row, a faithful representation of the run of the machine; this can be completed into a tiling of the entire plane if and only if the machine never halts — note that the tile at bottom right below corresponds to the machine entering the halt state, and no tile can fit beneath it. As the Halting Problem is undecidable, so too is the Completion Problem.

(On the other hand, note that it’s easy enough to tile in other ways, if we *don’t* place the seed tile. For example, we could just cover the plane with copies of the filler tile at upper right below.)



This example highlights the deep connection between *undecidability* and *computational universality*: The celebrated Church-Turing thesis in effect asserts that anything we might mean by computation can be realized by a Turing machine and thus by anything that can emulate a Turing machine. Wang’s Completion Problem is undecidable precisely because it has this property, precisely because completing a tiling from a

⁵It is easy enough, if one prefers, to use unmarked tiles that simply are required to fit together: we may convert the labels into geometric jigsaw-like bumps and notches: 

seed tile is “computationally universal” and can emulate *any* computation (albeit wildly inefficiently!).

As an aside, Wang posed the “Domino Problem” (or “Tiling Problem”): *Does a given set of tiles admit a tiling of the plane?* This is trivial for the sets constructed above since we may cover the plane with just copies of the blank filler tile. He noted that if the Domino Problem were in fact undecidable, there must exist sets of tiles that *do* admit tilings, but *none of which* are periodic — because just as we discussed in Section 1.1, if every set of tiles either does not admit a tiling or admits a tiling with a compact fundamental domain, then we have an algorithm for answering the Domino Problem: enumerate configurations, covering larger and larger disks, until we run out of possibilities (No, the tiles do not admit a tiling), or until we discover a fundamental domain (Yes, the tiles admit a tiling).

Wang reasonably conjectured no such *aperiodic* set of tiles could exist — after all, somehow, just by local rules, symmetry would have to be broken at all scales — but within a few years R. Berger, and then R. Robinson gave subtle proofs that the Domino Problem is undecidable, along the way producing aperiodic sets of tiles [2, 36]. Today, the Penrose tiles remain the most famous aperiodic set, but still, remarkably little is known about this phenomenon.

2.2 Fractran

John H. Conway’s Fractran [7], is an amusing generalization of the Collatz function described above. A Fractran program consists of a list of fractions, say these, discovered by D. Kilminster:

$$\frac{3}{11} \quad \frac{847}{45} \quad \frac{143}{6} \quad \frac{7}{3} \quad \frac{10}{91} \quad \frac{3}{7} \quad \frac{36}{325} \quad \frac{1}{2} \quad \frac{36}{5}$$

At each step, we will have some integer n ; we then multiply by the first fraction p/q in our list so that np/q is an integer, which we take as our integer at the next step. If no such fraction can be found, then the program halts, with output n .

So for example, beginning with 10, we would first multiply by $1/2$, obtaining 5; we then multiply by $36/5$, obtaining 36; in this manner, we see 858, then 234; then 5577; 1521; 3549; 8281; 910 and then 100; after another thirty-six steps, we have 1000; some one hundred fifty steps later, 10^5 , and three hundred four steps after that, 10^7 . In fact, this procedure generates the primes— every power of ten that appears is a prime power, and every prime power appears, in order! Astonishing!

Most generally, a Collatz-like function is of the form

$$f(n) = \left\{ \begin{array}{l} a_i n + b_i \quad \text{for } n \equiv i \pmod{N} \end{array} \right.$$

where N is some fixed counting number and all the a_i ’s and b_i ’s are rational, chosen in such a way that for integers n , $f(n)$ is an integer as well. The function f is completely specified by the list of values $N, a_0, b_0, \dots, a_{N-1}, b_{N-1}$ and our input is thus

Input: f and some integer n_0 .

Iterating f , obtaining, $n_0, f(n_0), f(f(n_0)), \dots$ we ask

Decision Problem: *Does iterating f on n_0 ever lead to a repeating loop of values?*

A Fractran program can be described as a Collatz-like function with all $b_i = 0$ (taking N to be the least common multiple of all the denominators in the program). Though the Fractran program above seems to work by magic, it is easy, just as with Wang’s Completion Problem, to see that this decision problem is undecidable, and that in fact, *any* computation can be encoded in Fractran!

Conway pulled off this trick by encoding Minsky register machines, which themselves are computationally universal [28, 29]. A Minsky register machine can be thought of as having “registers” a_1, a_2, \dots, a_k , each of which can take on a value in $0, 1, 2, \dots$, and a list of instructions of the form:

Instruction I_n : Increment a_n and go on to instruction I_{n_1} .

or

Instruction I_n : If register $a_n > 0$, decrement a_n and go on to instruction I_{n_1} ; otherwise go to instruction I_{n_2} .

It is not hard to see how we can build these up into subroutines and we do not gain any power by allowing more complex instructions such as

Instruction I : If $a > k$, decrement a by j , increment b by 2, and go to instruction A , otherwise increment a and go to instruction B .

Nor is it difficult to believe that anything we might be able to calculate using, say, assembly language, could be calculated by a Minsky register machine. (It is substantially more remarkable, as Minsky showed [28], that already just two registers are sufficient to carry this out!)

Given a such a list of instructions, it's not hard at all to construct a Fractran program that faithfully emulates the machine:

To each register a_n and each instruction I_n , we associate a unique prime number. For example, if our machine has three registers a, b, c and three instructions A, B, C we associate these with 2, 3, 5 and 7, 11, 13 respectively. Then at each step of the run, our integer will be of the form $2^a 3^b 5^c I$ where I is one of 7, 11, 13, depending on which instruction we are to read next.

An instruction of the form “**Instruction A : Increment a and go to instruction B** ” is encoded quite simply as the fraction 22/7: All of the other fractions in the program will have a factor of 11 or 13 in the denominator, but not 7, and so if at a given time our integer is $2^a 3^b 5^c 7$, we must faithfully execute instruction A , obtaining $2^{a+1} 3^b 5^c 11$. Similarly, an instruction of the form “**Instruction A : If register $a > 0$, decrement a and go to instruction B ; otherwise go to instruction C** ” is encoded simply as the pair of fractions 11/14 and 13/7, in that order.

There is only one small caveat: in a Fractran program, no instruction can jump to itself; for example, within an instruction A , we cannot go directly to A — the corresponding primes would cancel in the fraction encoding this instruction! But this is easy to work around, by introducing intermediate dummy instructions – we go to some instruction A' and then on to A . On the other hand, Fractran allows many shortcuts! It is quite pleasurable to work out just how Kilminster's program carries out its task; Conway has given other elegant examples in [7].

Precisely because arbitrary computations can be encoded as Fractran programs, problems such as these must be undecidable:

Input: *A finite sequence of rational numbers and a starting integer.*

We iteratively multiply by the first rational number in the sequence for which the result is an integer, unless no such rational is available and we halt; and ask:

Decision Problem: *Will we ever halt?*

Both this and the problem earlier in this section are just disguised forms of the Halting Problem.

2.3 Post Tag Productions

The example in Section 1.3 generalizes readily: specify an arbitrary alphabet \mathcal{A} ; for each letter $a \in \mathcal{A}$ a word σ_a which it produces; a starting word ω_0 ; and some fixed constant k .

At each step, we have a word ω_n ; if the length of this word is less than k , then we halt; otherwise, we produce ω_{n+1} by striking off the first k letters of ω_n and appending σ_a where a is the first letter of ω_n . We can write $ab\omega \rightarrow \omega\sigma_a$, where the variable b is any letter and ω is any word, to indicate this production rule.

For example, we might take $k = 2$, and $\mathcal{A} = \{a, b, c\}$, and specify $\sigma_a = cb, \sigma_b = aaa$ and $\sigma_c = a$. Taking $\omega_0 = aaa$, we produce in turn $aaa \rightarrow \cancel{aa}a cb = acb \rightarrow \cancel{ac}b cb \rightarrow baaa \rightarrow aaaaa$, which we abbreviate as a^5 . Focussing on just the words of the form a^n , we soon produce a^8, a^4, a^2 , and finally $a^1 = a$, at which point

we have too few letters to cross out, we cannot apply our rules and the process halts.

In fact, this system precisely encodes the Collatz function [10]! Beginning with a^n , with n even, it is not difficult to see that after n iterations, we obtain $a^{n/2}$. If $n > 1$ is odd, then $n + 1$ iterations produce $a^{(3n+1)/2}$. Our process eventually halts if and only if iterating the Collatz function eventually reaches 1.

Input: *A set of “tag” productions and a start word.*

Decision Problem: *Does the process eventually stop, beginning with the given start word?*

Tag production systems are a simple and useful undecidable system, with a beautiful mathematical history, initiated by Post while a student in the early 1920’s, reaching fruition with his lovely Normal Form Theorem in 1943 [33] and then the constructions of universal tag systems in the 1960’s [6, 28]. Y. Rogozhin [37] and others have used tag production systems to construct remarkably small “universal Turing machines” and M. Cook’s proof of the computational universality of Wolfram’s cellular automaton “Rule 110” relies on the device of “cyclic tag systems” [8].

Most generally, we may consider production systems of the following “canonical” form: Beginning with a finite list of words (“axioms”), we repeatedly produce new words (“assertions”), on the basis of old ones, by production rules of the form

$$\begin{array}{c} g_{11}\omega_{11}g_{12}\omega_{12}\dots\omega_{1n_1}g_{1(n_1+1)}, \\ g_{12}\omega_{12}g_{22}\omega_{22}\dots\omega_{2n_2}g_{2(n_2+1)}, \\ \vdots \\ g_{m1}\omega_{m1}g_{m2}\omega_{m2}\dots\omega_{mn_m}g_{m(n_m+1)} \\ \longrightarrow \\ g_1\omega'_1g_2\omega'_2\dots\omega'_ng_{n+1} \end{array}$$

where all the g ’s are specified, fixed, possibly empty words, and the ω ’s are variables, each ω'_i being some one of the ω_{jk} ’s. That is, each production is a rule for generating a new assertion on the basis of one or more earlier assertions.

As a specific example, on the alphabet $1, +, =$, take as a single axiom $1 + 1 = 11$ and production rules $\omega_1 + \omega_2 = \omega_3 \rightarrow 1\omega_1 + \omega_2 = 1\omega_3$ and $\omega_1 + \omega_2 = \omega_3 \rightarrow \omega_1 + 1\omega_2 = 1\omega_3$, producing such arresting assertions as $1111 + 11 = 111111$ and $1 + 11111 = 1111111$.

As a more interesting example, take alphabet $1, A, B, C, D$, axioms $A111, 11$ and productions

$$\begin{array}{lcl} A\omega & \rightarrow & A\omega 1 \\ A\omega 1 & \rightarrow & B\omega CD\omega 1 \\ \omega_1 1C\omega_2 1 & \rightarrow & 1\omega_1 C 1\omega_2 \\ \omega_1 BC\omega_2 1 & \rightarrow & B\omega_1 C\omega_2 1 \\ \omega_1 B\omega_2 1C\omega_3 D & \rightarrow & B\omega_1\omega_2 CD\omega_3 \\ 11BC\omega D 1 & \rightarrow & \omega 1 \end{array}$$

which generate the prime numbers! (That is, the assertions of the form $1 \dots 1$ are precisely the prime numbers in unary form.) [29] The active reader should be able to work out the mechanism behind this, by tracing out what is produced beginning from each $A1 \dots 1$, with either a prime or composite string of 1’s.

Post makes the point that essentially any formal, mechanizable system can be described as the applications of such rules: the local application of mechanical rules on strings of symbols. It is in fact not too difficult to design rules that can, say, emulate a Turing machine, or produce all the theorems under some first order formal system. But how simple can our systems be and still be powerful?

A production is in “normal form” if all of its production rules are of the very simple form $g\omega \rightarrow \omega g'$, with one variable and two fixed words; given the apparent simplicity of such systems, it is remarkable that:

Normal Form Theorem (Post, 1943) *Given any system in canonical form, on alphabet \mathcal{A} there exists a*

system in normal form on alphabet $\mathcal{A}' \supset \mathcal{A}$ so that the words produced by the first system are exactly those produced by the second, that contain only letters in \mathcal{A} .

In other words, in effect, every formal system can be captured by some system in normal form! But we can go further: Post's "tag"⁶ productions are a special kind system in normal form: in a tag system we require that for any pair of rules $g_1\omega \rightarrow \omega g'_1$, $g_2\omega \rightarrow \omega g'_2$ where both g_1 and g_2 begin with the same letter, then $g'_1 = g'_2$, and that all of the fixed words g on the left of the production rules have the same length. Such a system then can be thought of as in our initial example: at each step, cross off some k letters, tacking on a string on the end as determined by the first letter crossed out.

So in our initial example, as a normal form system, we have nine rules:

$$ax\omega \rightarrow \omega cb \quad bx\omega \rightarrow \omega aaa \quad cx\omega \rightarrow \omega a$$

where x is each of a, b, c .

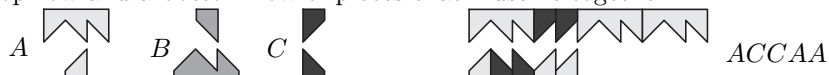
These systems are truly restricted and seem so simple! It seems incredible that they could have much power, yet by 1961, Minsky showed tag systems to be computationally universal [28], as Post suspected: given any Turing machine, there exists a tag system that completely encodes the machine's behavior. In particular, then, it is undecidable whether a given word is an assertion by a particular tag system, and it is undecidable whether a given tag system halts; Cocke and Minsky soon showed that $k = 2$ would suffice [6].

3 A few more examples

Fun, simple examples abound! Here are several more:

3.1 The Post Correspondence Problem

Can you solve the following puzzle? Pairs A, B, C of puzzle pieces are shown; select some sequence of pairs of pieces, forming a top row and a bottom row of pieces that must fit together.



It's not hard to see that any such sequence must begin with pair A , and continue $CCAA\dots$. It may be surprising that the shortest possible sequence requires seventy-five pairs:



There is actually a second solution of seventy-five pairs: Can you find it?

More generally, an instance of the Post Correspondence Problem has

Input: A finite collection of ordered pairs of words (A_i, B_i) in some fixed alphabet.

and we ask:

Decision Problem: Does the instance have a solution? That is, is there some sequence of indices i_1, i_2, \dots, i_n with $A_{i_1}A_{i_2}\dots A_{i_n} = B_{i_1}B_{i_2}\dots B_{i_n}$?

Post's original point was that it is quite easy to encode a given set of productions in Normal Form as an instance of the Correspondence Problem—certainly easy enough for the active reader to rediscover how!

⁶So called because Post envisioned a finite-state machine reading one portion of a tape and writing on another; as the length of the tape between them grows and shrinks, the read and write heads appear to be playing a game of tag.

In the example above, discovered by R. J. Lorentz and J. Waldmann [41], we could regard our alphabet as $\{0, 1\}$ (drawn as zags slanting up to the right and down to the right respectively), with pairs $(011, 0), (1, 011), (0, 1)$. This example is known to be the longest possible minimal solution known among instances with alphabets of two letters, in three pairs of words, each of length at most three. Heiko Stamer’s website *PCP@HOME* [41] features an evolving list of other record examples, all on an alphabet of two letters:

size	width	record instance known	# min sols	sol length	author
3	3	$((0,1),(1,011),(011,0))$	2	75	Lorentz & Waldmann
3	4	$((001,0),(1,1001),(0,01))$	1	452 (?) (> 340)	Rahn & Stamer
3	5	$((0,001),(00100,0),(10,0100))$	1	288	Rahn
4	3	$((0,011),(001,1),(1,00),(11,110))$?	595	Rahn
4	4	$((0,00),(0000,0101),(0001,10),(101,1))$	1	781	Rahn

M. Rahn [34] has identified many exotic instances, some of which may themselves beat these known records by substantial margins: Consider $((0,01),(1,10))$, which has no finite solution, but does admit the Thue-Morse sequence as an infinite solution. But adding one more pair of words gives an instance that has so far resisted analysis: $((0,01),(1,10),(0010,0))$.

It is impossible to explain any of these in any satisfying manner: *why* are they as they are — and why are others *not*? Apart from the Lorentz and Waldmann example, which is known to be a true record, it is highly likely that the actual records are far worse — and perhaps will never be known. In particular, as we soon shall see in Section 4.1, as the size and width increase, the largest possible minimum solution length must grow *faster than any function that can be explicitly described!*

This chart illustrates several hallmarks of undecidability: The interesting instances seem to have an arbitrary, *ad hoc* feel. The known record examples may or may not have much structure, though such structure may be needed in order to prove that the example actually has a solution. There are examples that seem beyond analysis, that will beat known records, if they do have a solution. The size of the true record solutions will grow rapidly with the size of the instances. Each of these phenomena is explainable by studying the “proof-complexity” of an undecidable decision problem, as we will discuss soon in Section 4.2.

On the other hand, it is not hard to prove the Post Correspondence Problem is undecidable: the essential idea is that any given Turing machine can be encoded as a collection of pairs of finite words so that any finite run of the machine corresponds to a finite sequence of pairs. The key is that the top row runs just one step ahead in time than the bottom row; the bottom row can “catch up” and there is a solution if and only if the machine eventually halts. Nicely written accounts of this proof appear in [29, 39]. More subtle constructions allow the encoding of an arbitrary Turing machine with as few as seven rules [31], or with only very short rules of width two [17]. It is not known whether instances with fewer than seven rules already have this power, but already the Collatz function can be encoded with just five — suggesting that this case is at a minimum very difficult to understand [34]. Up-to-date summaries can be found in [34, 41] and in the bibliographic references found therein.

3.2 NP-hard Problems

Thousands upon thousands of problems are now known to be NP-hard, and the threshold for this distinction does not seem particularly high: In essence, a problem is NP-hard if it is as difficult as any NP problem, that is, a problem with solutions that can be checked in a reasonably effective manner — below we touch cursorily on the elegant formalisms that make this rigorous, but among hundreds of excellent references, Garey and Johnson’s classic *Computers and Intractability* [15] and Sipser’s more recent textbook [39] will serve most beginning needs.

Yet undecidability lurks within every setting complicated enough to have NP-hard problems! Admittedly, the construction here will be somewhat artificial — to my knowledge, the following idea has not been exploited to produce simple, reasonable undecidable problems from NP-hard ones — but it is amusing to know that undecidability lurks in the Traveling Salesman Problem, the computer game Minesweeper, Sudoku puzzles, or any of the myriad NP-hard settings. And again, our main point is that undecidability resides in every

combinatorially interesting corner: here we give a large and robust collection of (o.k., seemingly contrived) examples.

We begin with S. Cook's original NP-hard problem, SAT, the satisfiability problem [9]:

Input: A collection of boolean variables $\{q_1, \dots, q_k\}$, and a collection of clauses, each of the form $p_1 \vee \dots \vee p_l$ with each $p_i \in \{q_1, \dots, q_k, \bar{q}_1, \dots, \bar{q}_k\}$.

Decision Problem: Is there an assignment of values to the variables so that all of the clauses are true — that is, is the instance satisfiable?

Cook's insight was that one can show a problem is NP-hard by showing it can encode finite runs of non-deterministic⁷ Turing machines. In particular, he showed SAT is NP-hard by giving a simple procedure for encoding each assertion that a given machine M can halt, accepting its input word ω , in less than n steps. The construction is strikingly elementary, with some similarity to the the tiling example of Section 2.1:

The variables encode such individual assertions as “the k th cell contains letter a at time t ”, or “the machine is reading the k th cell, in state s , at time t ”. The clauses build these up, averring “the machine is in exactly one state at time t ”, “the k th cell contains exactly one letter at time t ”, as well as statements that depend on the machine itself: “if at time t , the machine is in state s reading a in cell k , then at time $t + 1$ the machine is in state s' , one cell to the left (say) and cell k now contains letter b , or (as these encode non-deterministic machines) the machine is in state s etc.” Finally all the clauses assemble into one grand assertion: “The machine can halt and rest in the accept state by the end of the run, at time n .” This instance of SAT perfectly encodes the run of the machine— it is satisfiable if and only if the machine can indeed halt by time n .

It's not too difficult to work out a process to carry out this encoding and there will be a certain boring uniformity to all of the instances of SAT that arise from whatever encoding process we choose. Having a specific method in mind, then, consider the individual instances $SAT(M, n)$ of SAT encoding runs of each (deterministic) Turing machine M of length n on empty input, the instance being satisfiable if and only if the machine halts by time n . Fixing M , the instances $SAT(M, n)$ all appear roughly the same, the same building blocks repeated in a regular way, each instance in a sense contained within the next as n increases.

But then we have the following:

Input: A machine M .

Decision Problem: Is there an N such that for all $n \geq N$, the instances $SAT(M, n)$ are all satisfiable?

This is, of course, merely a disguised form of the Halting Problem and so is undecidable.

Now we leverage this out into the wider class of NP-hard problems. SAT itself is transparently in NP: any purported solution of an instance can be rapidly checked. Consequently, *any* NP-hard problem itself encodes SAT, and in turn encodes finite runs of arbitrary Turing machines. In particular, any such encoding must be comparatively simple and tractable (through the formalism of polynomial time reducibility) and just as with $SAT(M, n)$, the instances that arise will have a certain, rather tedious regularity, and can be quite explicitly described.

So, for example, HAM asks whether a given graph contains a Hamiltonian cycle; as HAM is NP-hard, we thus obtain an explicit means for constructing graphs $HAM(M, n)$ for each given machine M and counting number n , so that $HAM(M, n)$ contains a Hamiltonian circuit if and only if M halts by time n . Consequently it is undecidable, for a given M , whether infinitely many of the graphs $HAM(M, n)$ do in fact have a Hamiltonian circuit.

Or ... the list goes on: any of the thousands of problems known to be NP-hard may be converted into such an undecidable problem. Given an NP-hard decision problem P , we can construct for each given machine M and counting number n , an instance $P(M, n)$ so that $P(M, n)$ is decided **Yes** if and only if the machine M

⁷That is, a broader class of machines for which there might be more than one valid transition from a given state, reading a given symbol. Cook does not construct clauses asserting that the machine *does* halt in the accept state by time n , but that it *can*, that there is some sequence of valid transitions that leads to halting. For our discussion of decidability, we need only use deterministic machines, but the construction is indifferent and works either way.

halts by time n ; it is thus undecidable for each given M whether infinitely many of the $P(M, n)$ are decided **Yes**. And, again, as contrived as this construction appears, typically the specific instances of $P(M, n)$ will have a somewhat regular feel, having been built out of basic building blocks in a mechanical way — the collection is not *entirely* artificial.

3.3 Some very simple universal systems.

We close this section with some very simple, absolutely remarkable universal systems. Each of these is a single instance— a single Turing machine, for example — that by itself can emulate any computation, by reading and executing a program as input. Of course, the computer I am typing on now is exactly such a system (ignoring the limitations of time and memory!) but these are vastly simpler, though vastly less efficient!

Such very simple universal systems can be very useful in proofs that other problems are undecidable: one need only show that such a system can be encoded in the problem we are trying to analyze.

Word ladders were invented by Lewis Carroll: Can you, for example, convert the word **sleep** into the word **dream**, at each step changing one letter, always maintaining a legitimate English word?

Word ladders themselves are decidable, simply because there are only finitely many legitimate words to comb through. But the formal productions discussed in Section 2.3 are themselves a kind of generalization and there are many specific undecidable examples. G.S. Tsentin and Dana Scott found simple universal, undecidable examples in 1956 [38, 42]:

Input: *Two words α and ω written in the letters a, b, c, d, e*

We are allowed to make the following seven substitutions:

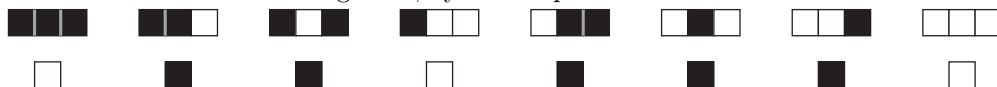
ac \leftrightarrow ca ad \leftrightarrow da bc \leftrightarrow cb bd \leftrightarrow db
ce \leftrightarrow eca de \leftrightarrow edb cdca \leftrightarrow cdcae

Decision Problem: *Is there a sequence of substitutions taking α to ω ?*

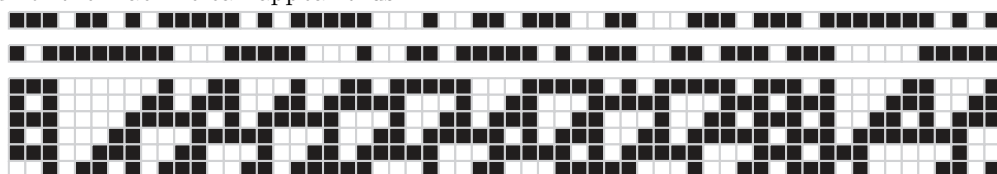
In 1966, Y. Matiyasevich [26] found a means of encoding any such system into one with just two letters and three substitutions! One of the substitutions, however, uses quite long words; encoding the example above requires a substitution between one word of length 304 and another of length 621.

Conway’s Game of Life is well known to almost every first semester programming student, as a fun and diverting homework assignment. It is less widely remembered that Conway specifically sought this as a simple, universal model of computation; one can “program” by specially setting the start state, and then recording how the playing field evolves through time [13].

Wolfram’s Rule 110 was conjectured to be universal and though litigation delayed publication for many years, M. Cook finally produced a proof [8] by showing this 1-dimensional cellular automaton can emulate “cyclic” tag systems, and that these themselves are computationally universal. This automaton has, at every time, an infinite row of cells each colored white or black. At the next step the color of each cell is determined by its current color and those of its neighbors, by the simple rule:



A short run of the machine can appear thus:



However, in order to emulate a given Turing machine, the automaton must begin on a specially prepared infinite but periodic pattern.

The (2, 3)-universal Turing machine was conjectured universal by S. Wolfram, who in 2007 offered a \$25,000 prize for a proof, awarded to A. Smith within just a few months:

ϕ	A	B
0	1RB	2LA
1	2LA	2RB
2	1LA	ORA

A vigorous discussion followed as Smith’s proof requires encoding the emulated machine as a particular infinite pattern on the tape. These patterns are not periodic, as with Rule 110, but they are highly ordered—that is, they are generated by simple machines that are not themselves computationally universal. As even a casual reader can guess, this result can be interpreted in a variety of ways: there is a widespread sense that though elegant and interesting, Smith’s proof enlarges the notion of “universal”. The interested reader can ask for no better starting point than the talk page of the relevant *Wikipedia* article [46], following the many outward links from there.

Many small universal machines working off of a finite input have been found by several authors, notably Yu.V. Rogozhin [37], C. Baiocchi [3], D. Woods and T. Neary [47].

4 What does this mean to the work-a-day mathematician?

By this time, the article has made its case as best it will: undecidable, computationally universal problems are ubiquitous, occurring everywhere in mathematics, even in the simplest settings. Why, indeed, even elementary arithmetic already has enough power to fully encode arbitrary computation, as Gödel, Turing and Kleene each point out [5, 20].

In essence all that is required is some sort of collection of simple elements, interacting in a relatively constrained manner; whether symbols on a page, mathematical objects in some structure, or a soup of chemicals, one expects the full power of computational universality to kick in readily. Computational universality can be found in devices made of Tinkertoys, or stone-age ropes and pulleys [11], or powered by billiard balls [12].

Of course there are many famous, natural examples in more classical mathematical realms. It is undecidable whether a diophantine equation has a solution; whether a given presentation describes the trivial group; whether two 4-manifolds are homeomorphic. Even in elementary analysis we have such trouble as [35]:

Let E be a set of expressions representing real, single valued, partially defined functions of one real variable, containing rational-valued constant functions, the identity function x , the functions $\sin x$, $\ln x$, $|x|$ and e^x , and closed under at least $+$, $-$, \times and composition (of course we restrict ourselves to just such a set in our algebra and calculus courses). Presume too there is at least one expression in E that has no antiderivative in E (for example, e^{-x^2}).

Input: An expression A in E

Each of the following is undecidable!

Decision Problem: (a) Is $A(x) = 0$ for all x ?

(b) Is $A(x) \geq 0$ for all x ?

(c) Does A have an antiderivative in E , a B in E with $B'(x) = A(x)$?

Would it relieve or upset our calculus students if they knew there were no way to decide whether a given elementary function has an elementary antiderivative?! In considering how much trouble a given setting might provide, we rely on

Conway’s Presumption: *If a lot is going on, everything can.*⁸

meaning that if a system has enough complexity, the betting man should assume there are enough building blocks to encode arbitrary computation. At the very least, the betting man is not likely to be contradicted! Examples abound, and it does not take much to lift off into computational universality — in some sense we might argue this is the generic condition!

4.1 How bad can these examples be?

Over and over again, we see very specific hallmarks of undecidability in the above examples: Suppose we have a collection of input instances I_0, I_1, \dots and a decision problem P . We ask: is there a mechanical procedure to decide P on input I_n ?

Let us suppose further that— just as with every example in this article— our problem is *semi-decidable*, that there’s a procedure that can decide in finite time at least if the answer is **Yes** but no procedure can answer in every instance that the answer is **No** (or, depending on which way round our problem is phrased, *vice versa*).

But even if we can decide in some instances, how long does this take? Fix some model of computation — Turing machines, Minsky register machines, a Java or Fractran program or any of the universal examples in this article — and some method of counting how many steps a given calculation takes. Then for any procedure for deciding P when we can, let us define $f(n)$ to be the minimum possible time it takes to find an answer, if we can find answer, and set $f(n) = 0$ if we cannot.

It doesn’t matter, really, how you do this. We have the remarkable:

Lemma: *The function f cannot be bounded by any computable function!*

Computable functions are simply those with some explicit description of how to calculate their values on the counting numbers: n , 2^n , n^n are all computable, and it is amusing to invent outrageous computable functions that defy imagination.⁹ Yet f beats all!

It’s not hard to see how; suppose f were bounded by some function g whose values we could compute. Then given n , compute $g(n)$ and then try to decide our problem P on input I_n ; if we try for $g(n)$ steps and have not yet succeeded, we know we never will and the answer must be **No**. But we know there can be no procedure that can always decide P and so no g can exist!

We snuck in an important point: this holds for *any* model of computation, and this article gives several. We can reinterpret this lemma in many, somewhat startling ways, such as:

- *Let $H(n)$ be the maximum Heesch number attained by any set of n tiles; then $H(n)$ cannot be bounded by any computable function.*
- *Among Fractran programs with n fractions that eventually halt, let $F(n)$ be the maximum number of steps required to do so; then $F(n)$ cannot be bounded by any computable function.*
- *Let $P(n, m)$ be the maximum length minimal solution of an instance of the Post Correspondence Problem of width m and n rules; then $P(n, m)$ cannot be bounded by any computable function.*

For Turing Machines, let $S(m, n)$ be the maximum number of steps an m -state machine on an alphabet of n -letters can run, starting on a blank tape, before halting; this too, of course, cannot be bounded by any

⁸Many people presume the same; Wolfram for example states something similar as his *Principle of Computational Equivalence*.

⁹For example, on the natural numbers, define $u(a, b, n) = a$, for $b = 1$; $u(a, b, n) = a^b$ for $n = 1$; and $u(a, b, n) = u(a, u(a, b-1, n), n-1)$ otherwise (thus $u(a, b, n) = a \uparrow \dots \uparrow b$ with $n \uparrow$'s in the Knuth arrow notation). Then set $v(1) = u(3, 3, 4)$ and take $v(n) = u(3, 3, v(n-1))$ for $n > 1$. The value $v(64)$ is the famous Graham’s Number! But we can consider such marvels as $V(n)$ equal to v composed with itself $V(n-1)$ times and other gems, playing this game all day, producing utterly incomprehensible but computable functions.

computable function and staggering lower bounds on S are actively being discovered: A few notable examples include [27]:

ϕ	A	B	C	D	E
0	1RB	1RC	1RD	1LA	1RH
1	1LC	1RB	OLE	1LD	OLA

Halts in 47,176,870 steps!
(Marxen, Buntrock, 1990)

ϕ	A	B	C	D	E	F
0	1RB	1LC	1LD	1LE	1LA	1LE
1	OLE	ORA	ORC	OLF	1LC	1RH

Halts after more than 2.584×10^{2869} steps!!
(T. and S. Ligocki, 2007)

In a span of just a few months in the winter of 2007-2008, T. and S. Ligocki produced a flurry of examples bounding $S(3, 3) \geq 119, 112, 334, 170, 342, 540$, $S(2, 5) > 1.9 \times 10^{704}$; $S(2, 6) > 2.4 \times 10^{9866}$; and $S(3, 4) > 5.2 \times 10^{13036}$. There is no reason to believe that the true values of S are not much much higher.

4.2 How hard are these examples to understand?

Examining specific instances of a specific undecidable problem, we see more and more outrageous behavior, leaping upwards with no computable bound. But that is hardly the worst of it. How difficult will it be to *prove* that these instances are as bad as we claim they are?

First, fix a model of computation; we've seen plenty in this article and any model will do, but for the sake of familiarity, imagine programming in some structured computer language such as C or Java.

Fix any consistent "reasonable" formal system, one that has a "reasonable" language, a "reasonable" concept of theorem and is "reasonably" powerful. An elementary, very broad discussion of these terms can be found in a lovely paper by A. Charlesworth [5], but in essence all we mean is that we can mechanically check that a given string of symbols is a statement in the system, that any statement or its negation (but not both) is to be true, that we can mechanically enumerate all possible proofs and check the soundness of any proof in the system, and that the system is powerful enough to prove such statements as *Program P halts after 100 steps* and *Program P halts eventually*. (Such statements are really just statements about basic arithmetic: for example, at each step of the run of a program on an electronic computer, we are just manipulating a gigantic binary number by simple rules of arithmetic, and we're only asking for a demonstration that we can make such a manipulation a finite number of times and then reach some desired result.)

But we can ask: What is the *shortest possible* proof of such an assertion, that is, what is the *proof complexity* $\pi(T)$ of each given theorem T in our system? Defining $\pi(n)$ to be the maximum proof complexity among all theorems of length n , we have the disturbing

Lemma: *The function π cannot be bounded by any computable function!*

That is, in any reasonable formal system, we expect short theorems with incredibly long proofs! This is really just a restatement of the lemma in the previous section, which in turn is a disguised form of the Halting Problem:

For any given specific program P , one of the two statements *Program P eventually halts* or *Program P does not ever halt* must be true. Let n be the length, in our formal system of the first statement and suppose $\pi(n)$ is bounded by some computable function $g(n)$. Then we can decide which of the two statements is true: Calculate $g(n)$ and then enumerate all proofs up to this length, checking to see if we've ever managed to prove the first statement. If at some point we have, we know the first statement is true. If we haven't, we never will, and so the second statement is true. In either case we will have decided which statement holds, but as the Halting Problem is undecidable, there can be no such g .¹⁰

But as before, this can be reinterpreted within any of the various models of computation we've discussed in this article. For example:

- *Mechanically enumerate all sets of tiles; let H_n be the n th set that does not admit a tiling; there is a*

¹⁰Kleene [5, 20] uses this approach to prove Gödel's Theorem: If, in our consistent, reasonable formal system, every *true* statement had a proof, then we can decide the Halting Problem: given a procedure P , just start enumerating proofs until we reach a proof of one of our two statements!

proof that it doesn't as discussed at the end of Section 2.1. Let $h(n)$ be the length of the shortest proof of this in our formal system. Then $h(n)$ cannot be bounded by any computable function!

- *Mechanically enumerate Fractran programs; let F_n be the n th set that eventually halts. We can prove this by simply running the program; let $f(n)$ be the length of the shortest proof of this in our formal system. Then $f(n)$ cannot be bounded by any computable function!*
- *Mechanically enumerate incidences of the Post Correspondence Problem; let P_n be the n th set that has a solution and let $p(n)$ be the length of the shortest proof that it does. Then $p(n)$ cannot be bounded by any computable function!*

That's just fun and games, but remember: these examples are stand-ins for a huge range of similarly computationally universal problems— this sort of trouble is everywhere. But this raises an essential issue:

To what extent are the ideas in this essay mathematical?

Don't misunderstand me— beyond a doubt, it is all mathematics: everything here can be proven to the usual acceptable standards. The theory of computation, with its foundational ties to the underpinnings of logic itself is as mathematical a subject as can be, with an abundance of beautiful and elegant proofs and constructions. And it's lovely that such simple ideas are sufficient to pin down some of the metamathematics, such as the asymptotic complexity of proofs. In this essay, we've only scratched the surface of this deep and beautiful topic, and we hope we have encouraged some readers to learn much more.

What of proofs that specific settings are computationally universal, or that specific problems are undecidable? Often these can be quite clever and appealing. It is certainly useful to know that certain problems — at least the especially important or natural ones — will forever be beyond full mathematical analysis (the word problem for groups comes to mind).

But what of studying individual instances of within a universal setting, looking at the specific growth of the functions discussed a moment ago, finding more and more outrageous examples of Busy Beaver Turing machine candidates, or high Heesch number sets of tiles? To what extent are *these* examples mathematical?

I must confess a delight in each of these examples— it's absolutely mesmerizing to see them in action. But not only do they leave one unsatisfied, with little means of penetrating just *why* these particular examples — but not others!— behave as they do, we can *prove* that we can *never* expect to understand why these, particularly, are just so.¹¹ That is, though there is some “reason” they are as they are, in the form of an astoundingly long proof, there cannot be any “good reason”, or understandable, short proof.

Mathematics — as is science — is fundamentally reductionist. Good mathematics synthesizes the disparate behavior seen in a wide range of examples into sweeping understanding. Mathematics, even difficult, highly technical mathematics, simplifies and unifies: It is no accident that we speak of “elegance”, or of “proofs in The Book.” Studying individual instances within undecidable problems, at least asymptotically, cannot be mathematical in this sense: no unified understanding will be available, whatever “proofs” there are giving no insight.

And yet we are surrounded! Such problems arise everywhere there is suitable combinatorial structure— and the bar is really quite low. These issues are likely to be increasingly hard to avoid as complexity and algorithms become more common tools within certain mathematical disciplines. And as many mathematicians begin to use computational experiments in their work, it is a natural temptation to explore increasingly intractable cases. In the sciences, too, “emergent” phenomena are increasingly studied, phenomena of precisely this sort in which small agents interact by combinatorial rules producing complex large scale structure;

¹¹Theology has not had a respectable place in the mathematics literature for many centuries; however we cannot resist pointing out a fundamental blasphemy inherent in the doctrine of “Intelligent Design” so fashionable in certain quarters: The word “design”, in a common sense, implies having some guiding principles, a simplified means of understanding the implications of choosing one set of conditions versus another. Within these computationally universal systems, by the arguments in this section, *no general design principles can exist*. That is, there is no simpler way to understand these implications than, well, just following them out. Now of course it would be foolish to presume just how an omnipotent deity would go about setting up a universe, life, etc. But to insist, as proponents of Intelligent Design do, that a deity *must* go about things in the most difficult, least powerful manner seems like a very limiting theology, to say the least.

mathematical or not, these systems are relevant, abundant and worth studying: It certainly seems wise to have some sense of where lies the edge of the abyss!

References

- [1] S.I. Adian and V.G. Durnev *Decision problems for groups and semigroups* Russian Math. Surveys **55** (2000), 207-296.
- [2] R. Berger, *The undecidability of the Domino Problem*, Memoirs Am. Math. Soc. **66** (1966).
- [3] C. Baiocchi *Three small universal Turing machines*, Machines Computations and Universality (M. Margenstern and Yu. Rogozhin, eds.) Springer Verlag (2001), 1-10.
- [4] E. Börger, E. Grädel, Yu. Gurevich *The Classical Decision Problem* Springer (2001).
- [5] A. Charlesworth *A Proof of Gödel's Theorem in Terms of Computer Programs*, Math. Mag. **54** (1981), 109-121.
- [6] J. Cocke and M.L. Minsky *Universality of tag systems with $P = 2$* J. Assoc. Comput. Mach. **11** (1964), 15-20.
- [7] J. H. Conway *FRACTRAN: A Simple Universal Programming Language for Arithmetic.*, Ch. 2 in *Open Problems in Communication and Computation* (Ed. T. M. Cover and B. Gopinath) Springer-Verlag (1987) 4-26.
- [8] M. Cook *Universality in Elementary Cellular Automata*, Complex Systems **15** (2004), 1-40.
- [9] S. Cook *The complexity of theorem proving procedures*. Proc. of the Third Annual ACM Symp. on Theory of Computing (1971) 151-158.
- [10] L. De Mol *Tag systems and Collatz-like functions* Theor. Comp. Sci. **390** (2008), 92-101.
- [11] A.K. Dewdney *The Tinkertoy computer and other machinations: computer recreations from the pages of Scientific American and Algorithm*, W.H. Freeman & Company (1993), New York.
- [12] E. Fredkin and T. Toffoli *Conservative Logic*, Int. J. Theor. Phys. **21** (1982), 219-253.
- [13] M. Gardner *The fantastic combinations of John Conway's new solitaire game "Life"*, Scientific American **223** (1971), 120123.
- [14] M. Gardner *Extraordinary nonperiodic tiling that enriches the theory of tilings*, Scientific American **236** (1977), 110-121.
- [15] M.R. Garey, and D.S. Johnson *Computers and Intractability: a guide to the theory of NP-completeness* W.H. Freeman and Co. (1979), New York.
- [16] B. Grünbaum and G.C. Shepherd *Tilings and patterns*, W.H. Freeman and Co. (1987).
- [17] V. Halava, T. Harjua, M. Hirvensalo, and J. Karhumäki *Post Correspondence Problem for short words*, Info. Proc. Let. **108**, 115-118.
- [18] D. Hofstadter *Gödel, Escher, Bach: an eternal golden braid* Vintage (1980).
- [19] J. Lagarias *The $3x + 1$ Problem: An Annotated Bibliography (1963 - 1999)*, arXiv:math/0309224v11.
- [20] S.C. Kleene *Recursive predicates and quantifiers*, Trans. Am. Math. Soc. **53** (1943), 41-73.
- [21] Lagarias, Jeffrey *The $3x + 1$ Problem: An Annotated Bibliography, II (2000-)*, arXiv:math/0608208v4.
- [22] C. Mann *Heesch's tiling problem*, Amer Math Monthly **111** (2004), 509-517.

- [23] C. Mann *The Edge-Marked Polyform Database* <http://www.math.utt Tyler.edu/polyformDB/>
- [24] M. Margenstern *Frontier between decidability and undecidability: a survey.*, Theor. Comp. Sci. **231** (2000), 217-251.
- [25] P. Michel and M. Margenstern *Generalized $3x + 1$ functions and the theory of computation*, preprint.
- [26] Yu. V. Matiyasevich *Simple examples of undecidable associative calculi*, Dokl. Akad. Nauk SSSR **173** (1967), 1264-1266; English transl., Soviet Math. Dokl. **8** (1967), 555-557.
- [27] P. Michel *Historical Survey of Busy Beavers*, <http://www.logique.jussieu.fr/~michel/ha.html>
- [28] M.L. Minsky *Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines*. Ann. of Math. **74** (1961) 437-455.
- [29] M.L. Minsky *Computation: finite and infinite machines* Prentice Hall (1967), Englewood Cliffs.
- [30] J. Myers *Polyomino, polyhex and polyiamond tiling*, <http://www.srcf.ucam.org/~jsm28/tiling/>
- [31] F. Nicolas *Post Correspondence Problem and semi-Thue systems*, lecture notes, arXiv:0802.0726v5.
- [32] T. Oliveira e Silva *Computational verification of the $3x+1$ conjecture* <http://www.ieeta.pt/~tos/3x+1.html>.
- [33] E. Post *Formal reductions of the combinatorial decision problem*, Am. J. Math. **65** (1943), 197-215.
- [34] M. Rahn *Entscheidbare Fälle des Postschen Korrespondenzproblems*, doctoral thesis, Universität Fridericiana zu Karlsruhe (2008).
- [35] D. Richardson *Some undecidable problems involving elementary functions of a real variable* J. of Symb. Logic **33** (1968), 514-520.
- [36] R.M. Robinson *Undecidability and nonperiodicity of tilings in the plane*, Inv. Math. **12** (1971), 177-209.
- [37] Yu.V. Rogozhin, *Small universal Turing machines* Theor. Comp. Sci. **168** (1996), 215-240.
- [38] D. Scott, *A short recursively unsolvable problem (abstract)*, J. Symbolic Logic **21** (1956), 111-112.
- [39] M. Sipser *Introduction to the Theory of Computation, 2nd. ed.* Course Technology (2005).
- [40] R. Smullyan *The Lady or the Tiger?: And Other Logic Puzzles Including a Mathematical Novel That Features Gödel's Great Discovery* Random House (1992).
- [41] H. Stamer *PCP@HOME* http://www.theory.informatik.uni-kassel.de/~stamer/pcp/pcpcontest_en.html
- [42] G. S. Tseitin *Associative calculus with insoluble equivalence problem*, Dokl. Akad. Nauk SSSR **107** (1956), 370-371. (Russian)
- [43] A. Turing *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc. Ser. 2 **42** (1936), 230-265.
- [44] H. Wang *Proving theorems by pattern recognition- II*, Bell System Technical Journal **40** (1961), 1-42.
- [45] S. Wolfram *New Kind of Science*, Wolfram Media (2002).
- [46] *Wolfram's 2-state 3-symbol turing machine*, talk page, *Wikipedia*.
- [47] D. Woods and T. Neary *On the time complexity of 2-tag systems and small universal Turing machines*, Proc. of the 46th Symp. on Found. Comp. Sci. (2006), 439-446.