There are several general ideas that are useful to organize computation, perhaps the two most important ones being:

- Recursion (self-similarity)
- Iteration (repetition)

Recursion is quite popular and directly supported in many programming languages.

Iteration usually requires some amount of extra work (and, to really make sense, strong support for functions).

**Definition**

Let $f : A \to A$ be an endofunction. The $k$th power of $f$ (or $k$th iterate of $f$) is defined by induction as follows:

$$f^0 = I_A$$
$$f^k = f \circ f^{k-1}$$

Here $I_A$ denotes the identity function on $A$ and $f \circ g$ denotes composition of functions.

Informally, this just means: compose function $f$ $(k-1)$-times with itself.

$$f^k = f \circ f \circ f \circ \ldots \circ f$$

$k$ terms
Without any further knowledge about \( f \) there is not much one can say about the iterates \( f^k \). But the following always holds.

**Lemma (Laws of Iteration)**

- \( f^n \circ f = f^{n+1} \)
- \( f^n \circ f^m = f^{n+m} \)
- \( (f^m)^n = f^{nm} \)

**Exercise**

Prove these laws by induction using associativity of composition.

Prof. Dr. Alois Wurzelbrunft stares at these equations and immediately recognizes a deep analogy to exponentiation.

He also remembers that there is a method for fast exponentiation based on squaring:

\[
a^{2^e} = (a^2)^e \]
\[
a^{2^{e+1}} = f \circ (a^2)^e
\]

which allows us to compute \( a^e \) in \( O(\log e) \) multiplications.

**Wurzelbrunft’s Conclusion:**

There is an analogous “fast iteration” method.

Good mathematicians see analogies between theorems or theories; the very best ones see analogies between analogies.

S. Banach

So is Wurzelbrunft brilliant?

Suppose we want to compute \( f^{1000} \). The obvious way requires 999 compositions of \( f \) with itself.

By copying the standard divide-and-conquer approach for fast exponentiation we could try

\[
f^{2^n} = (f^n)^2 \]
\[
f^{2^n+1} = f \circ (f^n)^2
\]

This seems to suggest that we can compute \( f^n(x) \) in \( O(\log n) \) applications of the basic function \( f \).

After all, it’s just like exponentiation, right?

If the function \( f \) in question is linear it can be written as

\[
f(x) = M \cdot x
\]

where \( M \) is a square matrix over some suitable algebraic structure. Then

\[
f^t(x) = M^t \cdot x
\]

and \( M^t \) can be computed in \( O(\log t) \) matrix multiplications.

So this is an exponential speed-up over the standard method, a clear case of computational compressibility.
Polynomial Maps

Another important case is when $f$ is a polynomial map

$$f(x) = \sum a_i x^i$$

given by a coefficient vector $a = (a_0, \ldots, a_1, a_n)$.

In this case the coefficient vector for $f \circ f$ can be computed explicitly by substitution. This is useful in particular when computation takes place in a quotient ring such as $\mathbb{R}[x]/(x^n - 1)$ so that the expressions cannot blow up.

Again, an exponential speed-up over the standard method, we have computational compressibility.

But Beware of Hasty Conclusions

But we cannot conclude that $f^t(x)$ can always be computed in $O(\log t)$ operations.

The reason fast exponentiation and the examples above work is that we can explicitly compute a representation of $f \circ f$, given the representation of $f$.

But, in general, there is no fast representation for $f \circ f$, we just have to evaluate $f$ twice.

Just think of $f$ as being given by a piece of $C$ code. We can produce another piece of $C$ code that computes $f^2$, and more generally for $f^t$, but the code just evaluates $f$ $t$-times, in the obvious brute-force way.

Exercise

Ponder deeply. Why is it foolish to expect all computations to be compressible? Think about complexity theory.

Digression: Hasty Conclusion I

Speaking about hasty conclusions, here is a simple inductively defined sequence of integers.

$$a_1 = 1$$

$$a_n = a_{n-1} + (a_{n-1} \mod 2n)$$

Thus, the sequence starts like so:

$$1, 2, 4, 8, 16, 20, 26, 36, 52, 60, 72, 92, 100, 110, 124, 146, 148, 182, 204$$

This seems rather complicated. The function appears to be increasing in a somewhat complicated manner.

Alas, there is a rude surprise.

Ultimately Linear

The sequence is ultimately linear: $a_{396+k} = a_{396} + k \cdot 194$ for $k \geq 0$.

The plot on the left is the sequence, on the right (in red) are the forward differences.

Exercise

Figure out why the sequence is ultimately linear.

Digression: Hasty Conclusion II

Here is another strange integer sequence:

$$a_n = \left\lceil \frac{2}{(2^{3/n} - 1)} \right\rceil - \left\lfloor \frac{2n}{\log 2} \right\rfloor$$

This time, the sequence starts like so:

$$0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \ldots$$

and continues like this for a long, long time, for trillions of terms.

Note that it is not so easy to compute the terms. At any rate, it sure looks like the sequence is constant $0$. Alas

$$0777 451 915 729 368 = 1$$

Iteration versus Recursion

Iteration can be construed as a special case of recursion.

$$F(0, y) = y$$

$$F(x+1, y) = f(F(x, y))$$

Then $F(x, y) = f^x(y)$.

Conversely, iteration can be used to express recursion. Suppose

$$f(0, y) = \pi(y)$$

$$f(x+1, y) = h(x, f(x, y), y)$$

is defined by primitive recursion.
As Iteration

Define a function $H : \mathbb{N} \times \mathbb{N} \times \mathbb{N}^k \to \mathbb{N} \times \mathbb{N} \times \mathbb{N}^k$ by

$$H(x, z, y) = (x + 1, h(x, z, y), y)$$

Then

$$\text{snd}(H^0(0, g(y), y)) = f(x, y)$$

Here snd is the projection onto the second component.

This is the natural definition, but if we wanted to we could make $H$ unary by coding everything up as a sequence number.

Unary Iteration

More precisely, suppose we have some simple basic arithmetic functions such as $x + y$, $x \times y$, $x^y$, $\text{rt}(x)$

Here $\text{rt}(x)$ is the integer part of $\sqrt{x}$. These suffice to set up coding machinery, which can then be used to replace recursion by iteration. It suffices to define functions via $f(x) = g^*(0)$ to get the same class as from the recursions above.

Exercise

Using clones, come up with a precise version of this statement and give a detailed proof.

Trajectories and Orbits

The trajectory or orbit of $a \in A$ under $f$ is the infinite sequence

$$\text{orb}_f(a) = a, f(a), f^2(a), \ldots, f^n(a), \ldots$$

The set of all infinite sequences with elements from $A$ is often written $A^\omega$. Hence the we can think of the trajectory as an operation of type

$$(A \to A) \times A \to A^\omega$$

that associates a function on $A$ and element in $A$ with an infinite sequence over $A$.

By the way . . .

Suppose $f : \mathbb{N} \to \mathbb{N}$ is computable.

Then for any $a \in \mathbb{N}$, the orbit set $A = \{ f^i(a) \mid i \geq 0 \}$ is semidecidable (the orbit is clearly recursively enumerable).

But in general, there is no reason for $A$ to be decidable: we cannot compute the stage $\sigma$ at which $b = f^\sigma(a)$.

Exercise

Find an example of a computable function with an undecidable orbit. How about the Ackermann functions $A_k$?

Terminology Warning

Sometimes one is not interested in the actual sequence of points but rather in the set of these points:

$$\{ f^i(a) \mid i \geq 0 \}$$

While the sequence is always infinite, the underlying set may well be finite, even when the carrier set is infinite.

In a sane world one would refer to the sequences as trajectories, and use the term orbit for the underlying sets. Alas, in the literature the two notions are hopelessly mixed up.

So, when we refer to a “trajectory” we will always mean the sequence, but, bending to custom, we will use “orbit” for both.

Digression: Dedekind’s Ketten (Chains)

Here is a clever definition due to Dedekind: given an endofunction $f$ and a point $a$, the corresponding chain is defined to be

$$\bigcap \{ X \subseteq A \mid a \in X, f(X) \subseteq X \}$$

Thus, the chain is the least set that contains $a$ and is closed under $f$. That is exactly the orbit of $a$ under $f$, considered as a set.

Who cares?

Dedekind’s definition does not require the natural numbers. In fact, it can be used to define them. In Dedekind’s view, this means that arithmetic can be reduced to logic.
Here is how. Suppose we have a function $f : A \to A$ and a point $a \in A$ such that

- $f$ is an injection,
- $a$ is not in the range of $f$,
- $A$ is the chain of $f$ and $a$.

Dedekind calls these sets **simply infinite**.

We can think of $a$ as $0$ and, more generally, we can think of $f^n(a)$ as $n$.

So this is a way of describing the natural numbers, the smallest infinite set, without any hidden references to the naturals.

At any rate, if the carrier set is finite, all trajectories must ultimately wrap around and all orbits must be finite:

What changes is only the length of the transient part and the length of the cycle (in the picture 6 and 10).

If $A$ is finite, then any orbit of $f : A \to A$ must be ultimately periodic:

$$f^t(x) = f^{t+p}(x)$$

for some $t \geq 0$, $p > 0$, which values depend on $x$.

**Definition**

The least $t$ and $p$ such that $f^t(x) = f^{t+p}(x)$ is the **transient length** and the **period length** of the orbit of $x$ (wrt. $f$).

Thus, an orbit is periodic iff the transient is 0.

Also, a function on a finite set has only transients of length 0 iff the function is injective iff it is a permutation.

According to Dedekind, the chain $C$ defined by $f$ and $a$ has the form

$$C = \bigcap \{ X \subseteq A \mid a \in X, f(X) \subseteq X \}$$

But note that $C$ is one of the $X$’s on the right hand side. So there is some (non-vicious) circularity in this approach. Most mathematicians would not bat an eye when confronted with definitions like this one, they are totally standard.

And the payoff is huge. For example, when Bernstein told Dedekind about his correct proof of the “Cantor-Schröder-Bernstein” theorem, he was shocked to hear that Dedekind had a much better proof, based on his chains.

The lasso shows the general shape of any single orbit, but in general orbits overlap. All orbits with the same limit cycle are called a **basin of attraction** in dynamics.
The Functional Digraph

As the last picture shows, it is natural to think of \( f \) as a directed graph on the carrier set where the edges indicate the action of \( f \).

Definition

The functional digraph (or diagram) of \( f : A \rightarrow A \) is defined as \( G_f = (A, E) \) where \( E = \{ (x, f(x)) \mid x \in A \} \).

Note that every vertex in \( G_f \) has outdegree 1, but indegrees may vary.

The non-trivial strongly connected components of the digraph are the limit cycles of the function. The weakly connected components are the basins of attraction.

Analyzing the Diagram

There are several natural parameters associated with the digraph that provide useful information about the function in question.

- **Indegree.** If all nodes have the same indegree \( k \) the function is \( k \)-to-1. Otherwise, determine the maximum/minimum indegree, the distribution of values.
- **Periods.** Count the number of limit cycles, and their length.
- **Transients.** Determine the length of the transients leading to limit cycles.

At least when the carrier set is finite we would like to be able to determine these parameters easily. Alas, even for relatively simple maps this turns out to be rather difficult.

Confluence (aka Basins of Attraction)

Definition

Let \( f \) be a function on \( A \) and \( a, b \in A \) two points in \( A \). Points \( a \) and \( b \) are confluent if for some \( i, j \geq 0 \):

\[
 f^i(a) = f^j(b).
\]

In other words, the orbits of \( a \) and \( b \) merge, they share the same limit cycle (which may be infinite and not really a cycle).

Reachability implies confluence but not conversely. For finite carrier sets reachability is the same as confluence iff the map is a bijection.

Confluence is an Equivalence

Proposition

Confluence is an equivalence relation.

Reflexivity and symmetry are easy to see, but transitivity requires a little argument.
Let \( f^i(x) = f^j(y) \) and \( f^k(y) = f^l(z) \), assume \( j \leq k \). Then with \( d = k - j \geq 0 \) we have

\[
 f^{i+d}(x) = f^{i+d}(y) = f^{l}(z).
\]

Each equivalence class contains exactly one cycle of \( f \), and all the points whose orbits lead to this cycle – just as in the last picture.

Small Example

Here is a somewhat frivolous operation on binary lists: given \( L \), replace the first element of \( L \) by 0, and then rotate to the left by 2 places.

Here is the orbit of a generic list (with symbolic entries) of length 6 under this operation:

<table>
<thead>
<tr>
<th></th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( x_5 )</th>
<th>( x_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
<td>( x_6 )</td>
</tr>
<tr>
<td>1</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
<td>0</td>
<td>( x_2 )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( x_5 )</td>
<td>( x_6 )</td>
<td>0</td>
<td>( x_2 )</td>
<td>0</td>
<td>( x_4 )</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>( x_2 )</td>
<td>0</td>
<td>( x_4 )</td>
<td>0</td>
<td>( x_6 )</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>( x_4 )</td>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>( x_2 )</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>( x_6 )</td>
<td>0</td>
<td>( x_2 )</td>
<td>0</td>
<td>( x_4 )</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>( x_2 )</td>
<td>0</td>
<td>( x_4 )</td>
<td>0</td>
<td>( x_6 )</td>
</tr>
</tbody>
</table>

So both transient and period are 3 in the generic case. But note the for special values of \( x_2, x_4 \) and \( x_6 \) the period may be shorter (ditto for the transient and \( x_1, x_3 \) and \( x_5 \)).

The following picture shows the behavior of all binary lists of length 6 under this operation.
There are two fixed points, and two 3-cycles.

\[ n = 8 \]

By Contrast \( n = 9 \)

Two Pictures, One Object

The two pictures represent the same functional digraph, albeit from a different perspective. Depending on what one is interested one or the other may be preferable.

Example: Exclusive Or

Here is a slightly more ambitious example, though the analysis turns out still to be fairly easy in this case. Consider the map

\[ f : 2^n \to 2^n \]

defined by

\[ f(x) = L(x) \text{ xor } R(x) \]

where \( L \) and \( R \) denote cyclic left- and right-shift, respectively, and \( \text{ xor} \) is bitwise exclusive or.

E.g., for \( n = 10 \) we have \( f(0,0,0,1,1,1,0,0,0,0) = (0,0,1,1,0,1,1,0,0,0) \)
Some Orbits, $n = 10$

A Battle Plan

How would one go about answering this question?

We need at the very least

- a program that takes as input an $n$-bit pattern and computes transient and period, and
- a program that takes as input $n$ and determines the whole diagram.

Why is the second item listed separately?

The Whole Diagram

The diagram is highly uniform in this case and can easily be described in terms of the general parameters.

- Every node has indegree 4.
- There are 40 limit cycles of length 6, 5 limit cycles of length 3 and one fixed point.
- The transient lengths for all points not on a limit cycle is 1.

Alas, this is not really interesting.

Real Question: How does this diagram depend on the parameter $n$?

The example above is just the special case $n = 10$.

Some Orbits, $n = 31$

Phase Space

Concretely, suppose you have a Boolean function $f : 2^{24} \to 2^{24}$

and you want to compute all transients and periods.

The space in question is $A = 2^{24}$, about 16 million elements.

Exercise

- What is a plausible algorithm?
- What if we had to deal with 32 bits instead, with 64?
We have already seen that iteration can produce very rapidly growing functions (much like recursion). Here is another example where iteration produces a rather perplexing result: every orbit ends in fixed point 0, though it looks like it should diverge towards infinity.

Suppose we write a number in base 2, say

\[ 266 = 2^8 + 2^3 + 2 \]

We can turn this into the complete binary expansion by writing the exponents also in base 2, and so on.

\[ 266 = 2^{2^3} + 2^{2+1} + 2 \]

where we really should write \(2^0\) instead of 1, but c’mon.

Now suppose we replace 2 in the representation everywhere by 3:

\[ 3^3 + 3^{3+1} + 3 \]

Unsurprisingly, this new number is much larger:

\[ 443426488243037769948249630619149892887 \approx 4 \times 10^{38} \]

Next, we write this number in complete base 3, and bump the base to 4. We get something like \(3 \times 10^{616}\).

Then we write this number in complete base 4 and bump to 5 . . .

Obviously, this process leads to a very rapidly increasing sequence of numbers. Now suppose we follow the base bump by subtracting 1, so the result will be a tiny little bit smaller than with a pure base bump. Call such a sequence a Goodstein sequence.

We expect Goodstein sequences to diverge since the base bump causes a huge increase, subtracting 1 should really not matter much. Alas . . .

Theorem

All Goodstein sequences converge to 0.

Alas, it is very hard to come up with good examples.

Starting at 3 we get the sequence

\[ 3, 3, 3, 2, 1, 0 \]

Starting at 4 we get the sequence

\[ 4, 26, 41, 60, 83, 109, 139, 173, 211, 253, 299, \ldots \]

It takes some \(10^{121,210,695}\) steps to get to 0!

The proof of Goodstein’s theorem uses ordinals and cannot be handled in Peano arithmetic. But note that the stopping time of these sequences is clearly computable: just do the arithmetic. Computable functions can be monsters.
A Termination Question

Here is a seemingly innocent question: Does the following program halt for all \( x \geq 1 \)?

```plaintext
while \( x > 1 \):
    // x positive integer
    if \( x \) even:
        \( x = x/2 \)
    else:
        \( x = 3 \times x + 1 \)
```

The body of the while-loop is rather trivial, just some very basic arithmetic and one if-then-else. This should not be difficult, right?

Collatz \( 3x + 1 \)

The Collatz Problem revolves around the following function \( C \) on the positive integers. There are several variants of this in the literature, under different names.

\[
C(x) = \begin{cases} 
1 & \text{if } x = 1, \\
\frac{x}{2} & \text{if } x \text{ even,} \\
\frac{3x + 1}{2} & \text{otherwise.}
\end{cases}
\]

This definition is slightly non-standard; usually case 1 is omitted and case 3 reads \( 3x + 1 \). Here are the first few values.

\[
\begin{array}{cccccccccccccc}
\hline
x & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & \ldots \\
C(x) & 1 & 1 & 5 & 2 & 8 & 3 & 11 & 4 & 14 & 5 & 17 & 6 & 20 & 7 & 23 & 8 & \ldots \\
\hline
\end{array}
\]

Arithmetic Only

The definition by cases for \( C \) is arguably the most natural.

But, we can get by without logic and make the arithmetic slightly more complicated. The following version does not treat argument 1 separately and does not divide by 2 when the argument is odd.

\[
C_{at}(x) = \frac{x}{2} - (5x + 2)((-1)^x - 1)/4
\]

Digression: The Name of the Game

The Collatz problem was invented by Lothar Collatz around 1937, when he was some 20 years old.

Since then, it has assumed a number of aliases:

Ulam, Hasse, Syracuse, Hailstone, Kakutani, …

Amazingly, in 1985 Sir Bryan Thwaites wrote a paper titled “My Conjecture” claiming fatherhood. Talking about ethics …

Boring Plot

Repetition

Of course, we are interested not in single applications of \( C \) but in repeated application. In fact, the Collatz program keeps computing \( C \) until 1 is reached, if ever.

Starting at 18:

18, 9, 14, 7, 11, 17, 26, 13, 20, 10, 5, 8, 4, 2, 1, 1, 1, …

Starting at 1000:

1000, 500, 250, 125, 188, 94, 47, 71, 107, 161, 242, 121, 182, 91, 137, 206, 103, 51, 256, 128, 64, 32, 16, 8, 4, 2, 1, 1, 1, 1, …

1367, 2051, 3077, 4616, 2308, 1154, 577, 866, 433, 650, 325, 488, 244, 122, 61, 182, 91, 137, 206, 103, 51, 256, 128, 64, 32, 16, 8, 4, 2, 1, 1, 1, …
Near Powers of 2

Starting at $2^{25} - 1 \approx 3.35 \times 10^7$.

It takes 282 steps to get down to 1.

Collatz Conjecture

More computation shows that for all $x \leq 3 \cdot 2^{53} \approx 2.7 \cdot 10^{16}$ the program always halts: $C$ reaches the fixed point 1. Many other values of $x$ have also been tested.

Based on computational evidence as well as various clever arguments one has the following conjecture:

Collatz Conjecture:
All orbits under $C$ end in fixed point 1.

Stopping Times

In an attempt to prove the Collatz Conjecture it is natural to try to investigate the stopping time: number of executions of the loop before 1 is reached.

$$\sigma(x) = \begin{cases} \min \{t \mid C^t(x) = 1\} & \text{if } t \text{ exists,} \\ \omega & \text{otherwise.} \end{cases}$$

So the Collatz Conjecture holds iff $\sigma(x) < \omega$ for all $x$.

The stopping time function $\sigma$ seems slightly more regular than $C$ itself, but it’s still rather complicated.
There clearly is some structure here (the dots are certainly not random), but what exactly is this mysterious structure?

A good way to make this precise is prediction: if we already know the first 5000 dots, how hard is it to predict the position of dot 5001?

At this point, no one knows how to do this without computing the whole orbit (at least not for arbitrary values of 5001).

Another potential way to gain insight in the behavior of the Collatz function is to plot the numbers in binary (time flows from left to right).

Could it be that the Collatz Conjecture is undecidable?

A priori, the answer is clearly No.

If we try to think of this as a decision problem, there is only one instance. But then there is a computable function that returns the right answer:

\[ \text{CC} \rightarrow \text{Yes} \]

or

\[ \text{CC} \rightarrow \text{No} \]

Alas, we don’t know which one works: this is a perfect example of a non-constructive argument.

And totally useless.
This problem is baked into our very definitions and rather difficult to deal with.

Suppose you have a decision problem \( \Pi \) where the set of instances \( I_\Pi \) is finite. Then \( \Pi \) is always decidable.

If \( |I_\Pi| = n \), there are \( 2^n \) potential algorithms, and one of them solves \( \Pi \). Too bad we often don’t know which one works.

Existence alone is really not good enough, we need something more constructive.

John Horton Conway, of Game-of-Life fame, found a beautiful way to show how undecidability lurks nearby.

Conway’s Idea:
How about constructing infinitely many Collatz Conjectures?

More precisely, come up with a family of functions that generalize the Collatz function slightly. Then ask if for one of these functions all orbits are ultimately periodic.

The classical Collatz function will be just one in this family, so understanding the whole family would of course solve the Collatz problem.

The hard part is to come up with a nice natural class of “Collatz-like” functions. Here is Conway’s approach: define

\[
C_n(a, b)(q \cdot k + r) = q \cdot a_r + b_r
\]

where \( a \) and \( b \) are two vectors of numbers of length \( k \) and \( 0 \leq r < k \). In other words,

\[
C_n(a, b)(x) = (x \text{ div } k) \cdot a_{x \text{ mod } k} + b_{x \text{ mod } k}
\]

The classical Collatz function is the special case

\( k = 2, \quad a = (1, 6), \quad b = (0, 4) \).

So now we have infinitely many functions to deal with (though one of them is perhaps more interesting than all the others).

A famous example of a Conway function other than the classical Collatz function is the following:

\[
T(2n) = 3n \\
T(4n + 1) = 3n + 1 \\
T(4n - 1) = 3n - 1
\]

This is just \( C_n(6, 3, 6, 3; 0, 3, 1, 2) \).

Proposition
\( T : \mathbb{N} \rightarrow \mathbb{N} \) is a bijection.

Exercise
Prove that the \( T \) function is a bijection. Then look for cycles under \( T \).

Looks very similar to the Collatz function.
But note that the lower line wobbles; there are really 3 linear functions here.
Known finite cycles are:

(1),
(2, 3),
(4, 6, 9, 7, 5),

Open Problem:
Are there any other finite orbits?
In particular, is the orbit of 8 finite?

This is a log-plot; it seems to suggest the orbit of 8 grows without bound. Of course, this is neither here nor there; maybe the values decrease after \( A(100, 100) \) steps, \( A \) the Ackermann function.

As we have seen, the notion of decidability is useless for problems with finite instance sets.

But there is another way one can try make sense out of the notion of a finite problem being “undecidable.”

- First, one fixes some formal system \( T \) (which is presumably adequate to talk about the problem at hand, something like Peano arithmetic or Zermelo-Fraenkel set theory).
- Then one shows that \( T \) can neither prove nor refute the claim that the problem at hand has a positive solution.

Classical example: the Continuum Hypothesis or the Axiom Of Choice in set theory.