CDM
(Semi) Decidability

Klaus Sutner

Carnegie Mellon Universality

30-decidability 2017/12/15 23:18

1. Solving Problems

- Decidability
- Diophantine Equations
- Generalized Collatz

Where Are We?

We have several ways to define the clone of computable functions:

- Register machines, Turing machines, Herbrand-Gödel, μ-recursive, λ-definable, while-computable, ...
- Turing has an excellent plausibility argument that we really have captured intuitive computability.

From now on, we just talk about computability and decidability

Time to analyze the basic properties of computability and decidability.

In TCS and math, the main interest in the machinery of computability comes from the desire to solve certain discrete problems. Here are the most important types:

Decision Problems Return a Yes/No answer.

Counting Problems Count objects of a certain kind.

Function Problems Calculate a certain function.

Search Problems Select one particular solution.

Definition

A decision problem consists of a set of instances and a subset of Yes-instances.

Problem:	**Primality**
Instance:	A natural number x.
Question:	Is x a prime number?

Here the set of all instances is $\mathbb{N}$ and the set of Yes-instances is $\{\, p \in \mathbb{N} \mid p \text{ prime} \,\}$.

The answer (solution) to any decision problem is just one bit (true or false).

Definition

A counting problem consists of a set of instances, each instance I is associated with a set of "solutions" sol(I). We need to calculate the cardinality of sol(I).

Problem:	**Prime Counting**
Instance:	A natural number x.
Solution:	The number of primes $p \leq x$.

Here the set of all instances is $\mathbb{N}$ and the "solutions" for x are $\{\, p \leq x \mid p \text{ prime} \,\}$. In the literature, this function is denoted $\pi(x)$.

The answer to a counting problem is always a natural number.

Definition

A function problem consists of a set of instances and, for each instance I, a unique solution $\mathsf{sol}(I)$.

Problem:	**Next Prime**
Instance:	A natural number x.
Solution:	The least prime $p > x$.

Instances are again $\mathbb{N}$ and the solution for $x \in \mathbb{N}$ is $\mathsf{sol}(x) = p$ where p is the appropriate prime (uniquely determined).

We insist that there always is a solution (otherwise sol would be a partial function).

Definition

A search problem consists of a set of instances and, for each instance I, a set of solutions sol(I).

In this case, "the" solution is not required to by unique. And, we allow for sol(I) to be empty. This turns out to be very convenient in practice.

Problem:	**Factor**
Instance:	A natural number x.
Solution:	A natural number z, $1 < z < x$, dividing x.

Note that the set of solutions is empty if x is prime.

- To solve a decision problem, a computable function has to accept each instance of the problem as input, and return "Yes" or "No" depending on whether the instance is a Yes-instance.
- To solve a counting problem, a computable function has to accept each instance x of the problem as input, and return the appropriate count $|\mathsf{sol}(x)|$ as answer.
- To solve a function problem, a computable function has to accept each instance x of the problem as input, and return the unique $\mathsf{sol}(x)$.
- To solve a search problem, a computable function has to accept each instance x of the problem as input, and return either an element of $\mathsf{sol}(x)$ or "No" if $\mathsf{sol}(x)$ is empty (the instance has no solutions).

Note that **Primality**, **Prime Counting** and **Next Prime** are closely connected: an algorithm for one problem can be turned into an algorithm for the other, plus a modest bit of overhead (a white lie). This idea of a reduction is critical in computability theory and in complexity theory.

Factor is a bit different, though: there are primality tests that provide no insight into factors of a composite number.

In fact, we now know that Primality is easy in the sense that there is a polynomial time algorithm for it, but we fervently hope that Factor will turn out to be hard (certain cryptographic methods will fail otherwise).

Our examples have $\mathbb{N}$ as the set of instances. In general, for some general decision problem Π we may have

- a set I_Π of instances, and
- a set $Y_\Pi \subseteq I_\Pi$ of Yes-instances.

For example, I_Π might be the collection of all ugraphs, and Y_Π could be the collection of all connected ugraphs.

If we code everything as naturals, then I_Π is always trivial (certainly p.r.). The same holds for the other types of problems.

More Instances

One might wonder what an algorithm is supposed to do with input that is not an instance of the problem in question.

There are two choices:

- We don't care: the behavior of the algorithm can be arbitrary.
- More realistic: the algorithm rejects bad input. This is perfectly reasonable since the collection of all inputs is always trivially decidable (in fact very low in the p.r. hierarchy).

  Something very fishy is going on if it is not.

Another issue is the choice of input data structure. Fortunately, in all practical cases there seems to be a canonical, natural choice. Essentially, all that is needed is:

- Natural numbers are given in binary.
- Nested lists of objects (hereditarily finite lists).

Note that numbers could actually be expressed as lists, but that seems a bit excessive (we might as well descend all the way into misery and use pure set-theory).

But recall that we ignore efficiency entirely at this point; we are only concerned with abstract computability. Things get trickier in the realm of low complexity classes.

Honest Instances (or: Don't Cheat)

In all our models the input must be given in explicit, unobfuscated form.

Not allowed are tricks like the following:

> Encode natural number n as $\langle 1, n \rangle$ whenever n is prime; and as $\langle 0, n \rangle$ when n is compound.

With this "input convention" primality testing would be trivial, but the coding procedure requires a lot of computational effort.

Worse, a similar trick could be used to trivialize any computational problem: just code the solution as part of the input.

Fuggedaboudit.

We will often informally talk about algorithms without offering any definition of this concept; every CS person knows what an algorithm is. We could cheat and say that an algorithm is just a computable function.

Alas, that is just not right. Certainly, every algorithm expresses a computable function, but it is surprisingly difficult to give a coherent and useful definition of what constitutes an algorithm. If you want to get an idea of how one might think about this problem, look at the paper by Moschovakis on the web.

You can think of an algorithm as a

> computable function, taking into account implementation issues and efficiency.

- Solving Problems

2. Decidability

- Diophantine Equations

- Generalized Collatz

Decision problems are slightly easier to deal with than the other types: the output is always 0 or 1.

Definition

A set $R \subseteq \mathbb{N}^k$ is decidable if the characteristic function char_R is computable.

A decision problem is decidable if the set of Yes-instances Y_Π is decidable.

Informally, this simply means that there is some algorithm that, given an instance of the problem, computes for a finite amount of time, and then returns the correct Yes or No answer. These algorithms are called decision procedures or decision algorithms.

Decision problems have been around since the day of the flood: one is interested in checking whether a number is prime, whether a polynomial is irreducible, whether a polygon is convex, and so on. Gauss certainly understood the computational difficulty of primality checking.

But the formal study of decision problems from a computational perspective is relatively new.

The first big splash came in 1900, when Hilbert presented his famous list of 23 open problems at the International Congress of Mathematicians in Paris.

Gottfried Leibniz (1646–1716)

In a way, the idea of decidability can be traced back to Leibniz's ars magna.

- ars inveniendi: generate all true scientific statements.
- ars iudicandi: decide whether a given scientific statement is true.

It is obvious that if we could find characters or signs suited for expressing all our thoughts as clearly and as exactly as arithmetic expresses numbers or geometry expresses lines, we could do in all matters insofar as they are subject to reasoning all that we can do in arithmetic and geometry. For all investigations which depend on reasoning would be carried out by transposing these characters and by a species of calculus.

Hilbert's list was enormously influential throughout the 20th century.

Some of Hilbert's Problems

1. Prove the Continuum Hypothesis. Well-order the reals.

2. Prove that the axioms of arithmetic are consistent.
   ...

8. Prove the Riemann Hypothesis.
   ...

10. Determination of the solvability of a Diophantine equation
    Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.

The Entscheidungsproblem is solved when one knows a procedure by which one can decide in a finite number of operations whether a given logical expression is generally valid or is satisfiable. The solution of the Entscheidungsproblem is of fundamental importance for the theory of all fields, the theorems of which are at all capable of logical development from finitely many axioms.

D. Hilbert, W. Ackermann
Grundzüge der theoretischen Logik, 1928

Note that Hilbert and Ackermann hedge a bit: the decision procedure for the Entscheidungsproblem needs to work only for areas where the fundamental assumptions can be finitely axiomatized.

That leaves open the possibility that some part of math might not be finitely axiomatizable, and thus outside of the reach of the Entscheidungsproblem.

Still, J. Herbrand pointed out that

> In a sense, the Entscheidungsproblem is the most general problem of mathematics.

No one really wanted to step into Hilbert's shoes in 2000, but some attempts were made to come up with a similar list of crucial problems for the 21st century.

1. Prove the Riemann Hypothesis.
2. Prove the Poincaré Conjecture. Solved, Perelman 2002/3
3. Is $\mathbb{P} = \mathbb{NP}$?
4. Bound the number of integer roots of a Diophantine polynomial.

   ...

18. What are the limits of intelligence, both artificial and human?

The interesting point here is that a surprising number of these problems have a strong computational perspective. Note that Steve Smale is a classical mathematician (proved the Poincaré Conjecture for dimensions $n \geq 5$, Fields medal 1966), but has become very interested in computational issues lately.

Computer science has effectively invaded all other sciences.

That's great from a CS perspective, but it means additional work: a CSist is supposed to be able to communicate effectively with scientists from other fields. This is harder than you may think.

Lemma

The decidable sets are closed under intersection, union and complement. In other words, the decidable sets form a Boolean algebra.

Proof.

Consider two decidable sets $A, B \subseteq \mathbb{N}^k$. We have two Turing machines M_A and M_B that decide membership.

Idea: Run both M_A and M_B on input x, returning output b_A and b_B where $0 \leq b_A, b_B \leq 1$.

For intersection return $\min(b_A, b_B)$,

for union return $\max(b_A, b_B)$,

for the complement of A return $1 - b_A$.

□

As usual, we have used elementary arithmetic to express logical operations.

conjunction	$\min(b_A, b_B)$	$b_A \wedge b_B$
disjunction	$\max(b_A, b_B)$	$b_A \vee b_B$
negation	$1 - b_A$	$\neg b_A$

The Boolean algebra of decidable sets is effective: we can represent the elements by natural numbers, and the corresponding algebraic operations are computable (even primitive recursive).

Here is an innocent question:

> Is every decision problem $A \subseteq \mathbb{N}$ decidable?

If this were true we could construct, for any $A \subseteq \mathbb{N}$, a Turing machine M_A that, on input any number $x \in \mathbb{N}$, halts with output $\text{char}_A(x)$.

Unfortunately, the answer is **NO**.

Here is a cardinality argument due to Cantor that shows that there are lots of undecidable problems.

Theorem

There are undecidable decision problems.

Proof.

There are uncountably many subsets of $\mathbb{N}$ (to be precise: $2^{\aleph_0}$).

But there are only countably many Turing machines (recall our coding machinery).

Hence uncountably many problems $A \subseteq \mathbb{N}$ are not decidable.

□

Note that this argument, like others in set theory, leaves a bitter after-taste: it does not produce any concrete undecidable problem. As usual, there is a more constructive approach.

Problem:	**Halting**
Instance:	Index $e \in \mathbb{N}$.
Question:	Does TM M_e halt on input e?

A slightly more precise way to express this would be to fix a universal Turing machine $\mathcal{U}$ and to ask whether

$$\mathcal{U}(e, e) \downarrow$$

The choice of the UTM is entirely arbitrary, but will not affect the decidability status of Halting.

A standard diagonalization argument shows that Halting cannot be solved by a Turing machine.

Theorem

The Halting Problem is undecidable.

Again, we should say that Halting is undecidable for $\mathcal{U}$, but since this works for all UTMs, there is no point in specifying a particular machine.

There is no UTM that halts on all even indices, and diverges on all odd indices.

Diagonalization was invented by Cantor in the context of set-theory. It is rather surprising that the entirely non-constructive cardinality argument due to Cantor also turns out to be the main tool in establishing results in the computational universe.

As it turns out, there is another notion, closely related to decidability, that is arguably even more important, more later.

Definition

A set is semidecidable if there is a Turing machine that, on input x, halts if x belongs to the set, and fails to halt otherwise.

We will call this a semidecision procedure. Obviously, decidable implies semidecidable.

You can think of this as a broken decision algorithm: if the answer is Yes, the algorithm works properly and stops after finitely many steps. But, if the answer is No, it just keeps running forever, it never produces a result.

Note that Halting is a natural example of a semidecidable problem: we can determine convergence in finitely many steps, but divergence takes forever.

Later we will establish the critical connection between decidability and semidecidability:

Lemma

A set is decidable iff the set and its complement are both semidecidable.

Pick any model of computation: Herbrand-Gödel, μ-recursive, λ-definable, Turing-computable, register machine computable.

As Turing has shown, there is a universal "machine" $\mathcal{U}$ which can be used to produce an effective enumeration $(\{e\})_e$ of all computable functions:

$$\{e\}(x) \simeq \mathcal{U}(e, x)$$

Exercise

Figure out what the universal "machine" would look like in the other models.

We can exploit this to obtain a uniform representation for all computable functions. This works particularly well for Kleene's μ-recursive functions.

We claim that there is a primitive recursive relation $T(e, x, t)$ and a primitive recursive function D (in fact, both T and D are quite straightforward) such that

$$\{e\}(x)\downarrow \iff \exists t\, T(e, x, t)$$

$$\{e\}(x) \simeq D(\min\big(\, t \mid T(e, x, t)\,\big)$$

$T(e, x, t)$ essentially means: the computation of $\mathcal{U}$ on e and x terminates after at most t steps.

Kleene's T Predicate

$T(e, x, t)$

- e the index of a Turing machine M
- x an input argument for M
- t a witness for a halting computation of M on input x.

The witness is usually the (sequence number that codes) a sequence of configurations $C_0, C_1, \ldots, C_n$ of M.

Since we have an alleged witness, T is easily decidable and primitive recursive.

Moreover, given the right t, it is easy to read off the output of the computation (that's D's job).

The standard interpretation of the witness t is that it codes the whole computation (what exactly that is depends on the model).

Alternatively, we can think of $t \in \mathbb{N}$ as just time, the number of steps the computation takes.

This is just as good, because there is a primitive recursive function W such that

$$W(e, x, t) = \text{ actual full computation of length } t$$

So where does undecidability come in?

The reason Kleene's T fails to yield the decidability of halting is that we cannot bound the search for t in any computable fashion: given $M = M_e$ and x, there is no way to predict the number of steps in the computation of M on x, we have to run the full computation (computational incompressibility).

Note that this is not a problem in any real algorithm: given some particular input we can always compute ahead of time an upper bound on the running time of the algorithm. Ditto for memory requirement. Proof: check all the algorithms in 451.

Still Unhappy?

Halting may well seem like a somewhat unsatisfactory example of an undecidable problem: it's a perfect case of navel gazing. Certainly it does not deal with a question at least superficially unrelated to computability.

Actually, anyone who has ever written code in a language like C would have to admit that Halting is really quite natural.

But how about undecidable problems that are of independent interest? Perhaps something that was studied even before the concept of an algorithm was invented?

- Solving Problems
- Decidability

3. Diophantine Equations

- Generalized Collatz

Perhaps the most famous example of an undecidability result in mathematics is Hilbert's 10th problem (HTP), the insolubility of Diophantine equations.

A Diophantine equation is a polynomial equation with integer coefficients:

$$P(x_1, x_2, \ldots, x_n) = 0$$

The problem is to determine whether such an equation has an integral solution.

Theorem (Y. Matiyasevic, 1970)

It is undecidable whether a Diophantine equation has a solution in the integers.

The proof is too complicated to be presented here, but the main idea is a reduction:

> *Show that decidability of Hilbert's 10th problem implies decidability of the Halting problem.*

More precisely, call a set $A \subseteq \mathbb{Z}$ Diophantine if there is a polynomial P with coefficients over $\mathbb{Z}$ such that

$$a \in A \iff \exists\, x_1, \ldots, x_n \in \mathbb{Z} \left(P(a, x_1, \ldots, x_n) = 0 \right).$$

This condition is rather unwieldy, it is usually fairly difficult to show that a particular set is in fact Diophantine.

Even numbers: $a = 2x$.

Non-zero: $ax = (2y - 1)(3z - 1)$.

Naturals (Lagrange): $a = x_1^2 + x_2^2 + x_3^2 + x_4^2$.

Closure under intersection (sum of squares) and union (product).

Again: Semidecidability

It turns out that exactly the semidecidable sets are Diophantine. In particular, the Halting Set is Diophantine, and so it must be undecidable whether an integer polynomial has an integral solution.

It is clear that every Diophantine set is semidecidable: given a, we can simply enumerate all possible $\boldsymbol{x} \in \mathbb{Z}^n$ in a systematic way, and compute $P(a, \boldsymbol{x})$.

If we ever hit 0, we stop; otherwise we run forever.

Surprisingly the opposite direction also holds, but this is much, much harder to show.

First M. Davis was able to show that every semidecidable set A has a Davis normal form: there is a polynomial such that

$$a \in A \iff \exists\, z\, \forall\, y < z\, \exists\, x_1, \ldots, x_n \left(P(a, x_1, \ldots, x_n, y, z) = 0\right).$$

Davis, Putnam and Robinson then managed to remove the offending bounded universal quantifier at the cost of changing P to an exponential polynomial (containing terms x^y).

Lastly, Matiyasevic showed how to convert the exponential polynomial into an ordinary one.

Note that if we could produce a computable bound on the size of a possible root we could use brute-force search to determine whether one exists (in principle, in reality we die an exponential death).

For example, it is known that if

$$y^2 = x^3 + c$$

has a solution, then it has a solution bounded by $\exp((10^{10}|c|)^{10000})$.

So it would not be unreasonable to think that integer roots of $P(x_1, \ldots, x_n)$ can be bounded by some rapidly growing but computable function of n, the degree d of P, and the largest coefficient c. Remember Friedman's self-avoiding words?

Perhaps something insanely huge like

$$A\left(n!\,|c|^d, n^{n^{d+17}}\right)$$

would work?

If not, try an even higher level of the Ackermann hierarchy.

Alas, finding bounds even in very concrete cases turns out to be quite difficult. Try to find a positive solution for the Pell equation:

$$x^2 - 991y^2 - 1 = 0.$$

Of course, $x = 1$, $y = 0$ is a trivial solution. The smallest positive solution here is

$$\begin{aligned} x &= 379516400906811930638014896080 \\ y &= 12055735790331359447442538767 \end{aligned}$$

An old puzzle, supposedly due to Archimedes, about the size of the herd of cattle owned by the sun god, comes down to solving a system of linear equations and then the Pell equation

$$x^2 - 4102864232784 24\, y^2 - 1 = 0.$$

In this case, the least positive solution has 103273 and 103265 decimal digits.

Exercise (in futility)

Try to find a positive solution to $313(x^3 + y^3) = z^3$.

Note that the choice of $\mathbb{Z}$ as ground ring is important here. We can ask the same question for polynomial equations over other rings R (always assuming that the coefficients have simple descriptions).

- $\mathbb{Z}$: undecidable
- $\mathbb{Q}$: major open problem
- $\mathbb{R}$: decidable
- $\mathbb{C}$: decidable

Decidability of Diophantine equations over the reals is a famous result by A. Tarski from 1951, later improved by P. Cohen.

One Implication

It is true that an algorithm for HTP over $\mathbb{Z}$ would produce an algorithm for HTP over $\mathbb{Q}$.

Consider $P(\boldsymbol{x}) = 0$ with $\boldsymbol{x} \in \mathbb{Q}$. This is equivalent to

$$\exists\, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{Z}\left(P(\boldsymbol{y}/\boldsymbol{z}) = 0 \wedge z_1 \ldots z_k \neq 0\right)$$

Of course, this is exactly the wrong direction.

Exercise

Why does undecidability over $\mathbb{Z}$ not simply imply undecidability over $\mathbb{Q}$? What is the obstruction?

Since we can encode arbitrary semidecidable sets as Diophantine equations, we can in particular encode universal sets.

That means that there is a single polynomial with parameter a for which the question

$$\exists\, x_1, \ldots, x_n\, P(a, x_1, \ldots, x_n) = 0$$

is already undecidable.

As one might suspect, there is a trade-off between the degree d of such a polynomial and its number of variables n. Here are some known (d, n) pairs that admit universal polynomials:

$$(4, 58), (8, 38), (12, 32), \ldots, (4.6\,10^{44}, 11), (8.6\,10^{44}, 10), (1.6\,10^{45}, 9)$$

A Prime Polynomial

$$(k+2)(1-[wz+h+j-q]^2-[(gk+2g+k+1)(h+j)+h-z]^2-$$
$$[16(k+1)^3(k+2)(n+1)^2+1-f^2]^2-[2n+p+q+z-e]^2-$$
$$[e^3(e+2)(a+1)^2+1-o^2]^2-[(a^2-1)y^2+1-x^2]^2-$$
$$[16r^2y^4(a^2-1)+1-u^2]^2-[n+l+v-y]^2-[(a^2-1)l^2+1-m^2]^2-$$
$$[ai+k+1-l-i]^2-[((a+u^2(u^2-a))^2-1)(n+4dy)^2+1-(x+cu)^2]^2-$$
$$[p+l(a-n-1)+b(2an+2a-n^2-2n-2)-m]^2-[q+y(a-p-1)+$$
$$s(2ap+2a-p^2-2p-2)-x]^2-[z+pl(a-p)+t(2ap-p^2-1)-pm]^2)$$

This polynomial P has 26 variables and degree 25. $P(\mathbb{N}^{25})\cap\mathbb{N}$ is the set of prime numbers.

Suppose P is an integer-coefficient polynomial of $2n$ variables.

There are two players: Peter (parameters) and Ursula (unknowns). They alternate picking natural numbers $a_1, x_1, a_2, x_2, \ldots, a_n, x_n$. If, in the end,

$$P(a_1, \ldots, a_n, x_1, \ldots, x_n) = 0$$

Ursula wins, otherwise Peter wins.

Clearly, given P, it is undecidable whether Peter or Ursula has a winning strategy.

But things are actually much worse; even if Ursula has a winning strategy, it may not be constructively describable.

Consider the Diophantine game

$$(x_1 + a_2)^2 + 1 = (x_2 + 2)(x_3 + 2)$$

So Peter has only one choice, but Ursula has three.

The RHS is always composite, but the LHS could be prime.

Indeed, if there are infinitely many primes of the form $k^2 + 1$, then Peter has a winning strategy; otherwise Ursula wins.

Alas, the existence of these primes is an open problem in number theory.

- Solving Problems
- Decidability
- Diophantine Equations

4. Generalized Collatz

A Termination Question

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

```
while x > 1:                    // x positive integer
  if x even:
      x = x/2
  else:
      x = 3 * 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?

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.

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

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

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
$C(x)$	1	1	5	2	8	3	11	4	14	5	17	6	20	7	23	8	...

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.

$$C_{\text{ari}}(x) = x/2 - (5x + 2)((-1)^x - 1)/4$$

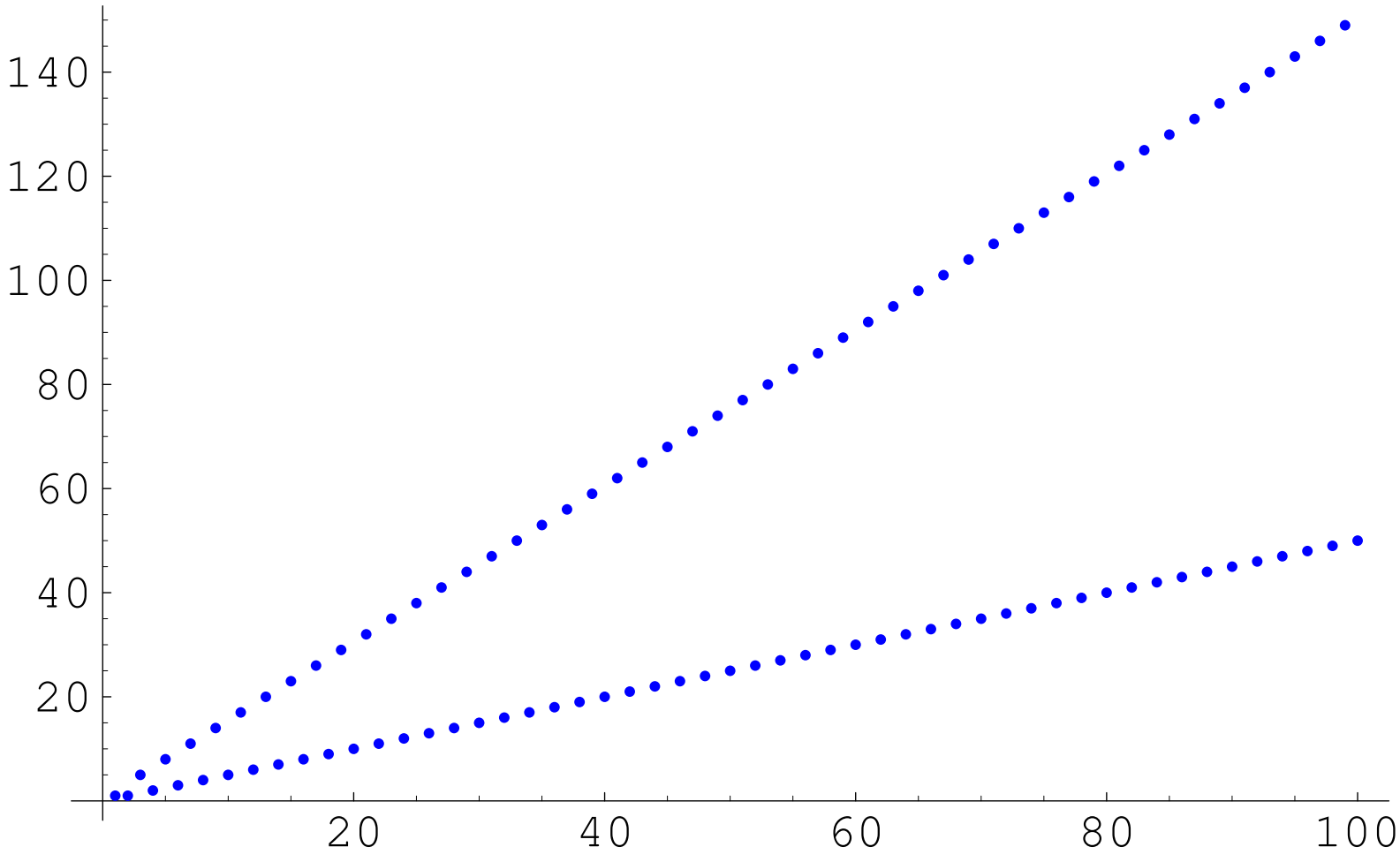
The Collatz problem was invented by Lothar Collatz around 1937, when he was some 20 years old.

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



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:

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

Starting at 1000:

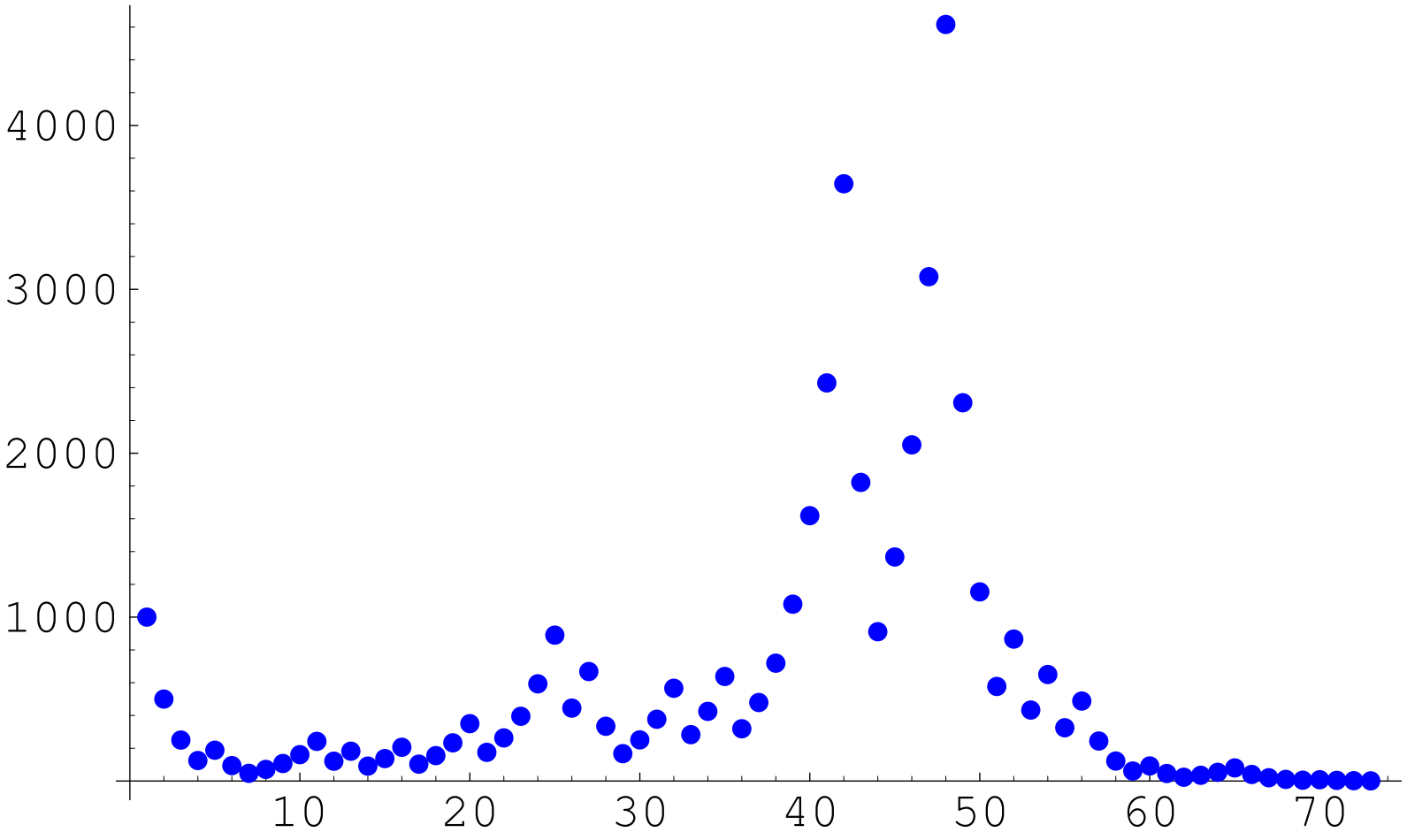
1000,500,250,125,188,94,47,71,107,161,242,121,182,91,137,

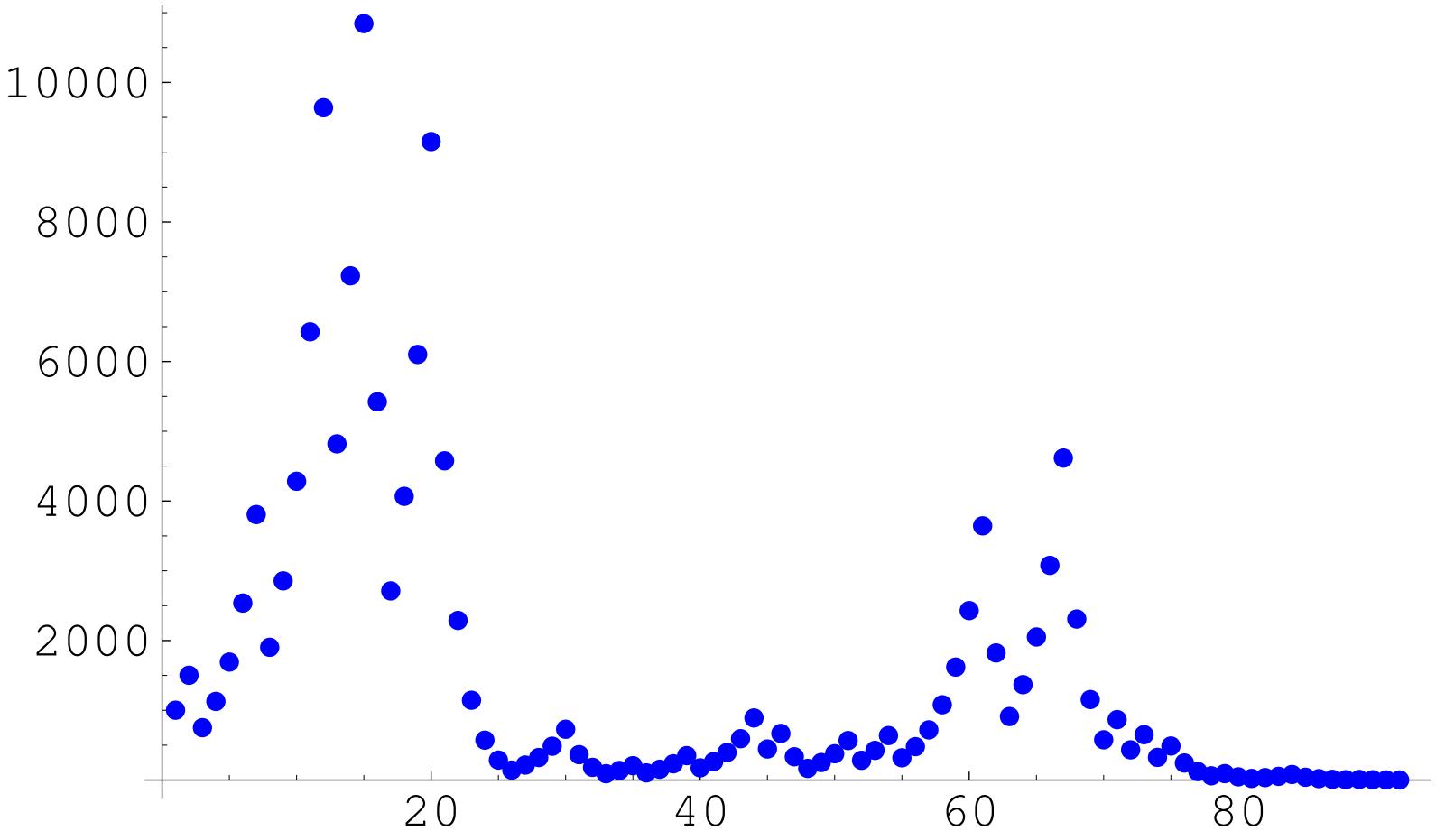
206,103,155,233,350,175,263,395,593,890,445,668,334,167,251,

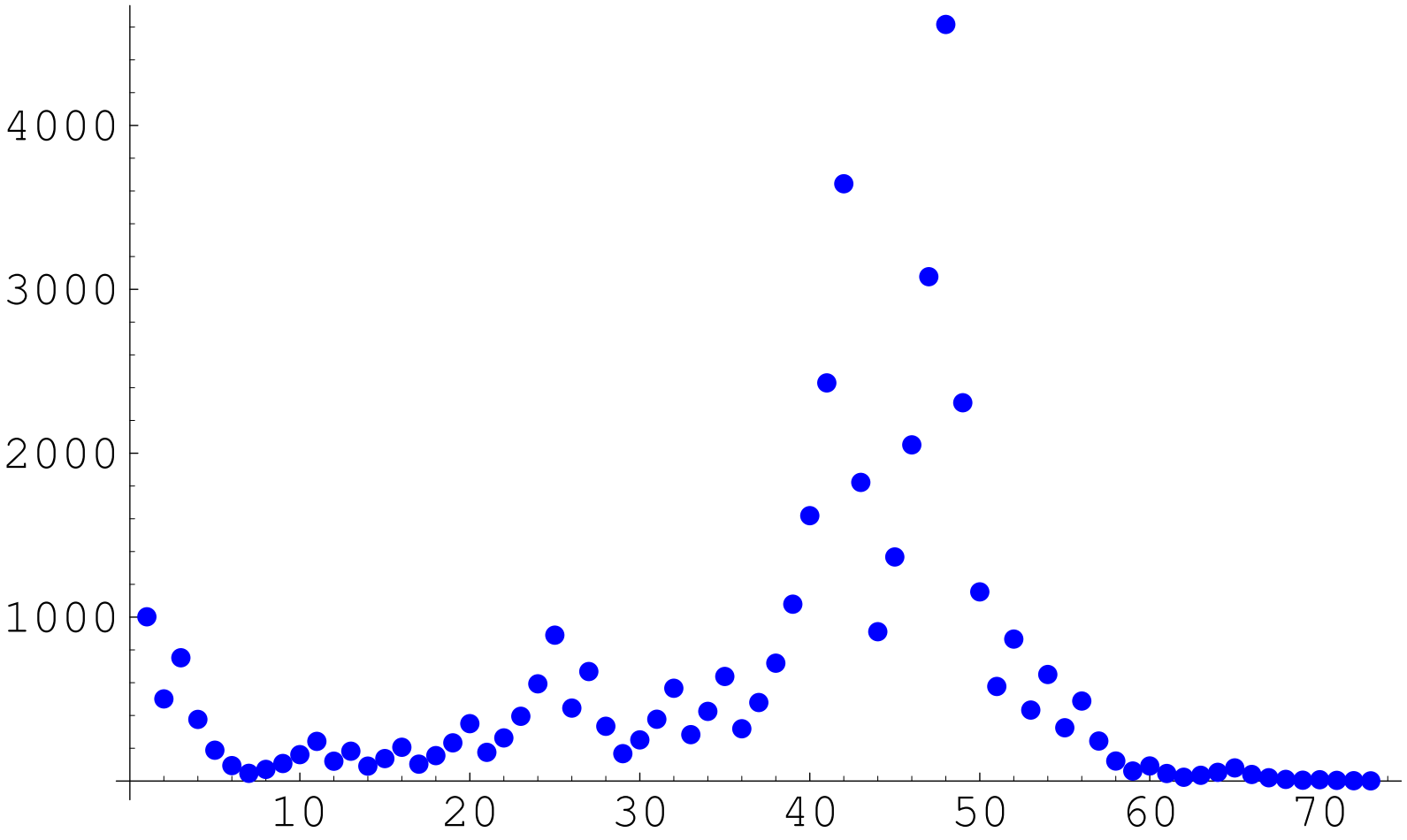
377,566,283,425,638,319,479,719,1079,1619,2429,3644,1822,911,

1367,2051,3077,4616,2308,1154,577,866,433,650,325,488,244,

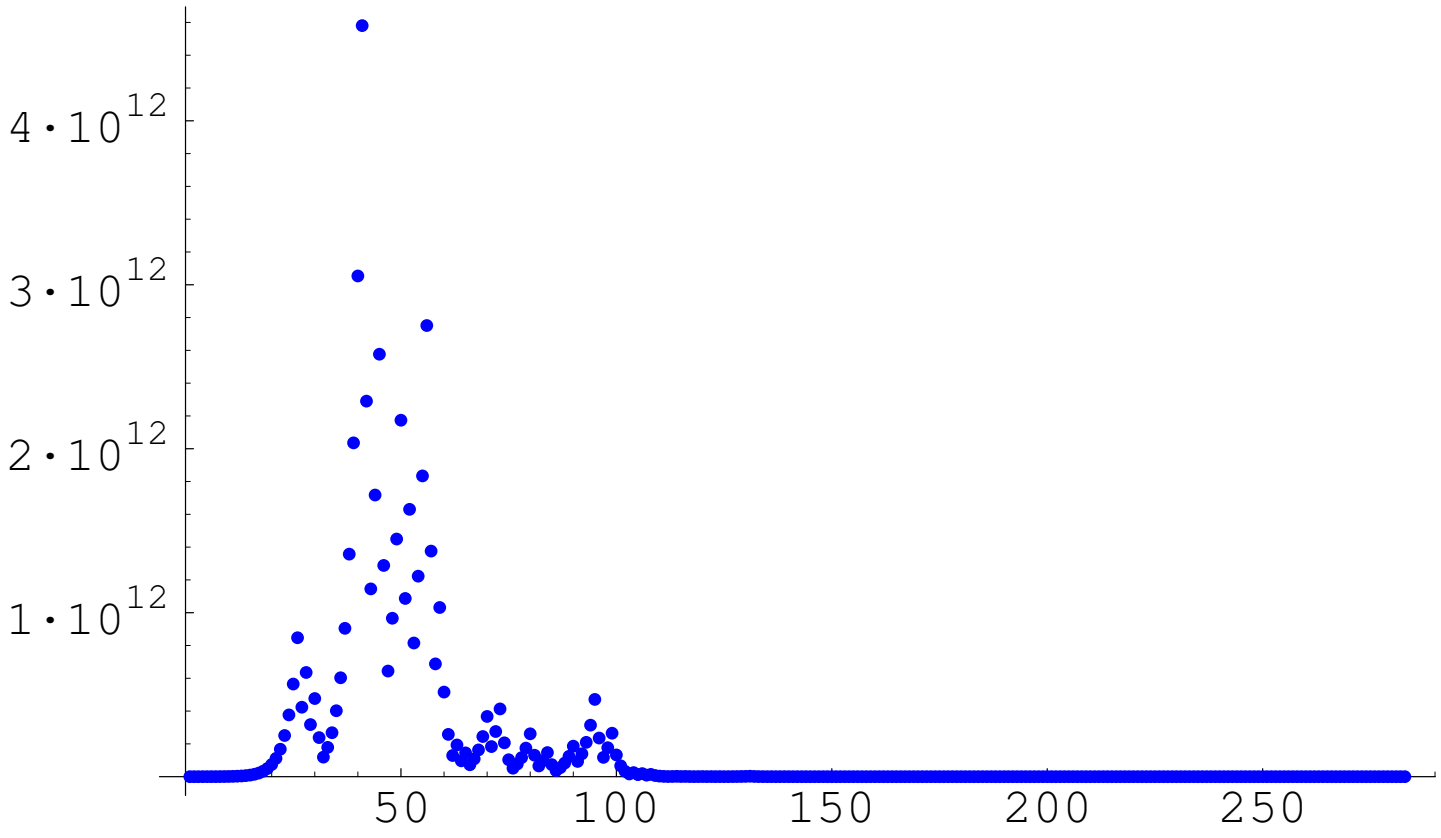
122,61,92,46,23,35,53,80,40,20,10,5,8,4,2,1,1,1,1,...


4000
3000
2000
1000
10
20
30
40
50
60
70


10000
8000
6000
4000
2000
20
40
60
80


4000
3000
2000
1000
10
20
30
40
50
60
70

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



It takes 282 steps to get down to 1.

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.

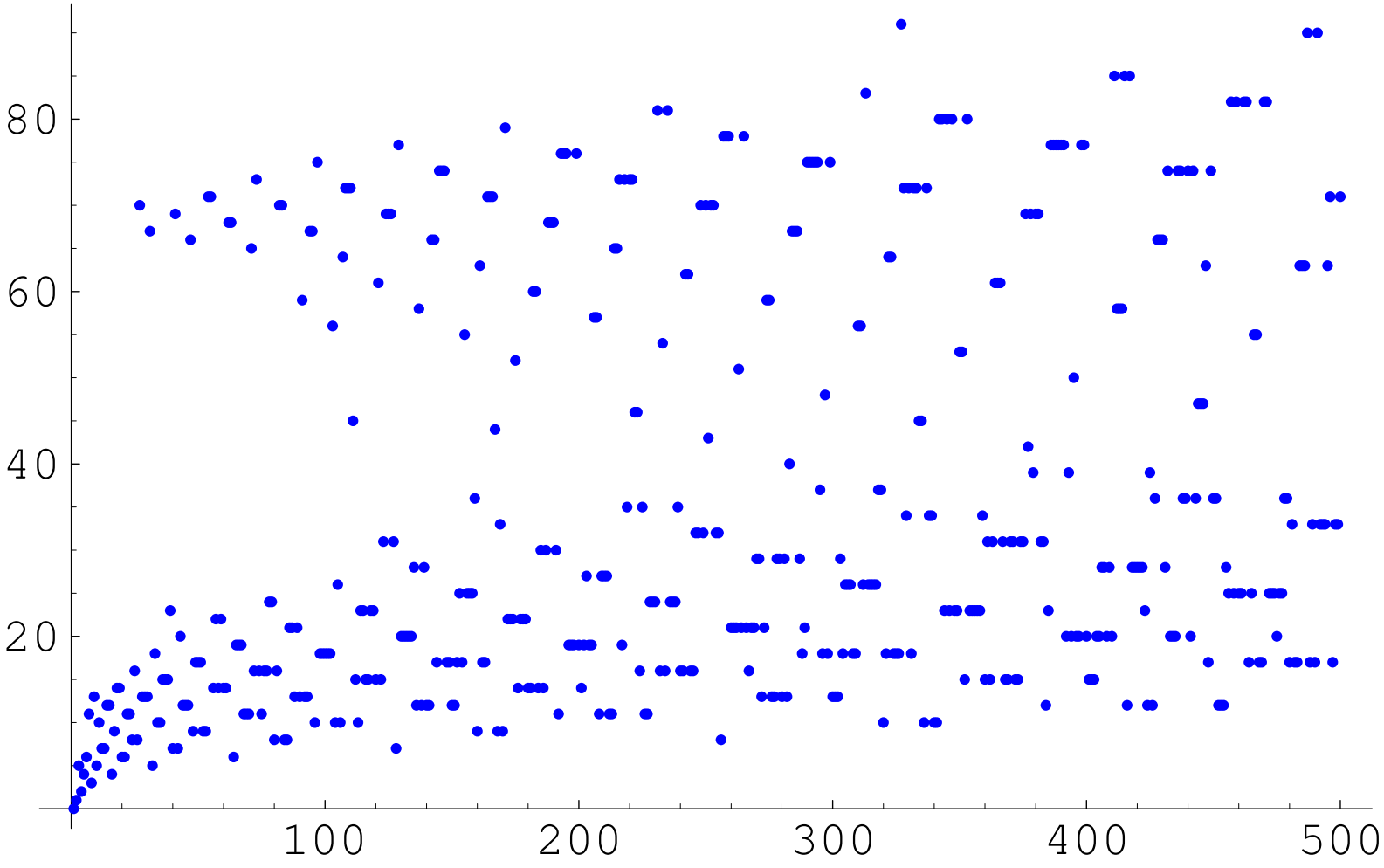
In an attempt to prove the Collatz Conjecture it is natural to try to investigate the stopping time: number of executions of the loop before 1 is reached.

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

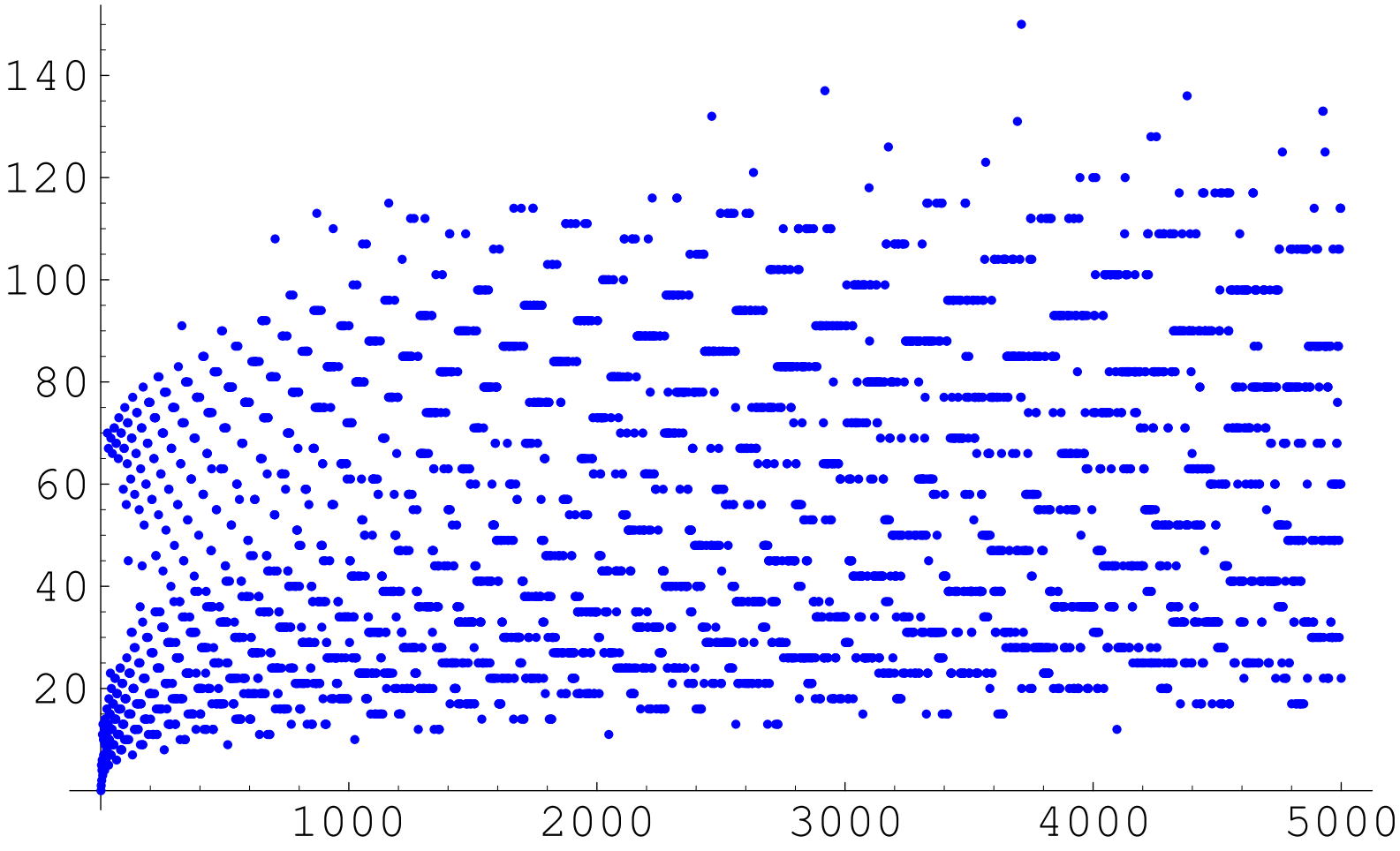
So the Collatz Conjecture holds iff $\sigma(x) < \infty$ for all x.

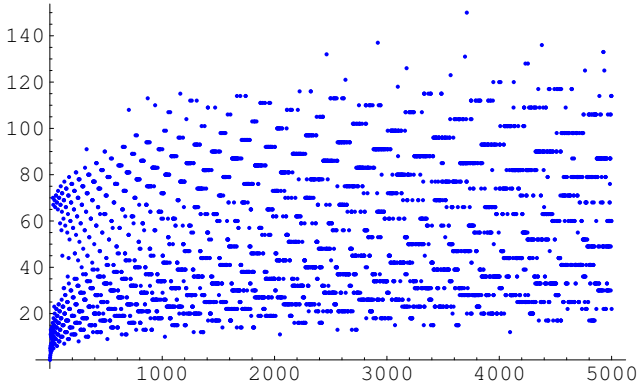
The stopping time function σ seems slightly more regular than C itself, but it's still rather complicated.

Stopping Times up to 500



Stopping Times up to 5000





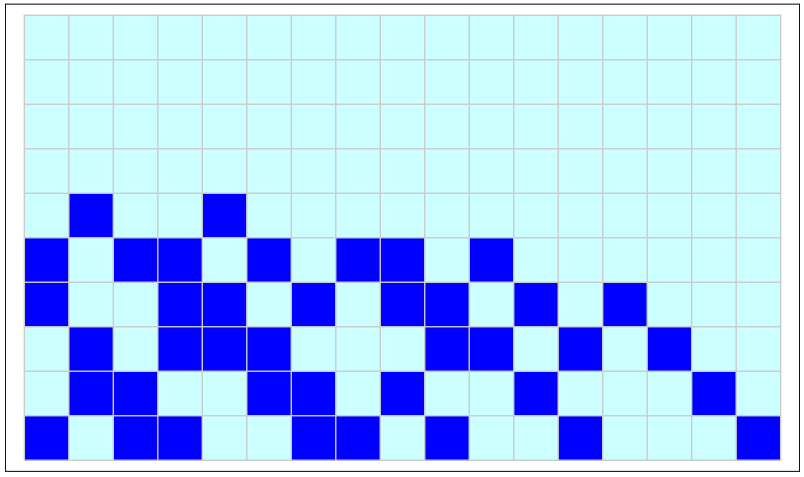
There clearly is some structure here (the dots are certainly not random), but what exactly is this mysterious structure?

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?

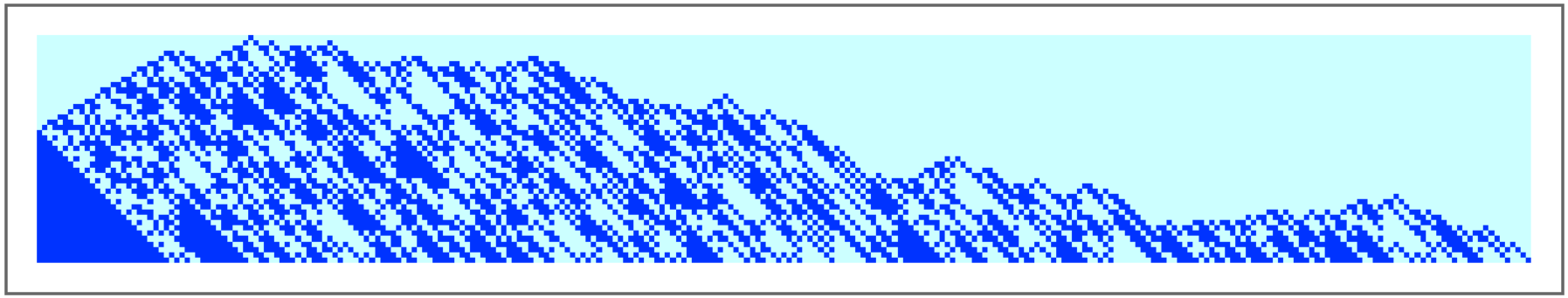
At this point, no one knows how to do this without computing the whole orbit (at least not for arbitrary values of 5001).

Orbit of 25 in Binary

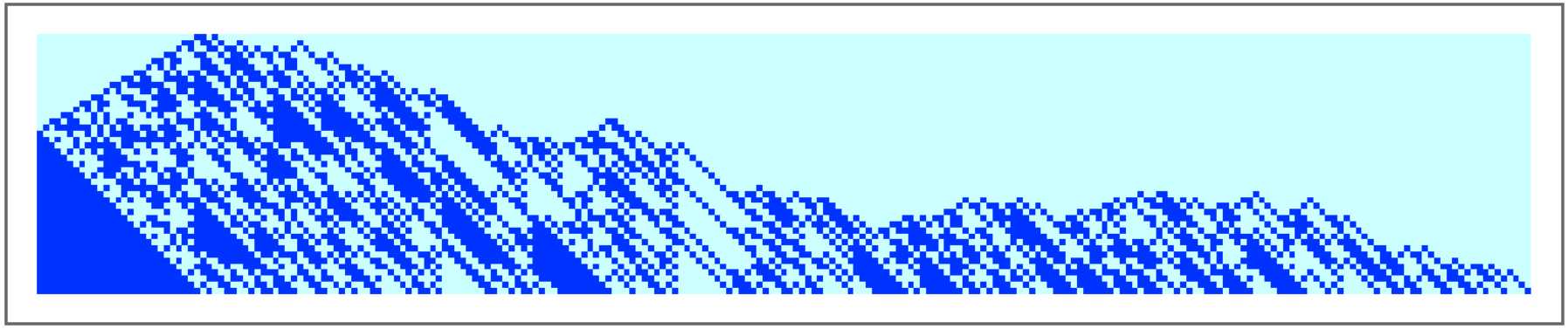
Another potential way to gain insight in the behavior of the Collatz function is to plot the numbers in binary (time flows from left to right).



$x = 2^{25} - 1 \approx 3.35 \times 10^7$



$x = 2^{27} - 1 \approx 1.34 \times 10^8$



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:

$$\text{CC} \mapsto \text{Yes}$$

or

$$\text{CC} \mapsto \text{No}$$

Alas, we don't know which one works. Totally useless.

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

Suppose you have a decision problem Π where the set of instances I_Π is finite.

Then Π is always decidable.

If $|I_\Pi| = n$, there are 2^n potential algorithms, and one of them solves Π. 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

$$\mathrm{Cn}(\boldsymbol{a}, \boldsymbol{b})(q \cdot k + r) = a_r \cdot q + b_r$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are two vectors of numbers of length k and $0 \leq r < k$.

The classical Collatz function is the special case

$$k = 2, \boldsymbol{a} = (1, 6), \boldsymbol{b} = (0, 4).$$

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

Problem:	**Conway-Collatz Problem**
Instance:	The parameters $\boldsymbol{a}$, $\boldsymbol{b}$.
Question:	Is every orbit of $\mathrm{Cn}(\boldsymbol{a}, \boldsymbol{b})$ ultimately periodic?

Theorem

The Conway-Collatz Problem is undecidable.

It remains undecidable even if all the b_i's are 0.

The theorem indicates that there is no good general way to answer questions about Collatz-like functions. So it is not entirely surprising that the classical Collatz function is also very difficult to analyze.

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

$$T(2n) = 3n$$
$$T(4n+1) = 3n+1$$
$$T(4n-1) = 3n-1$$

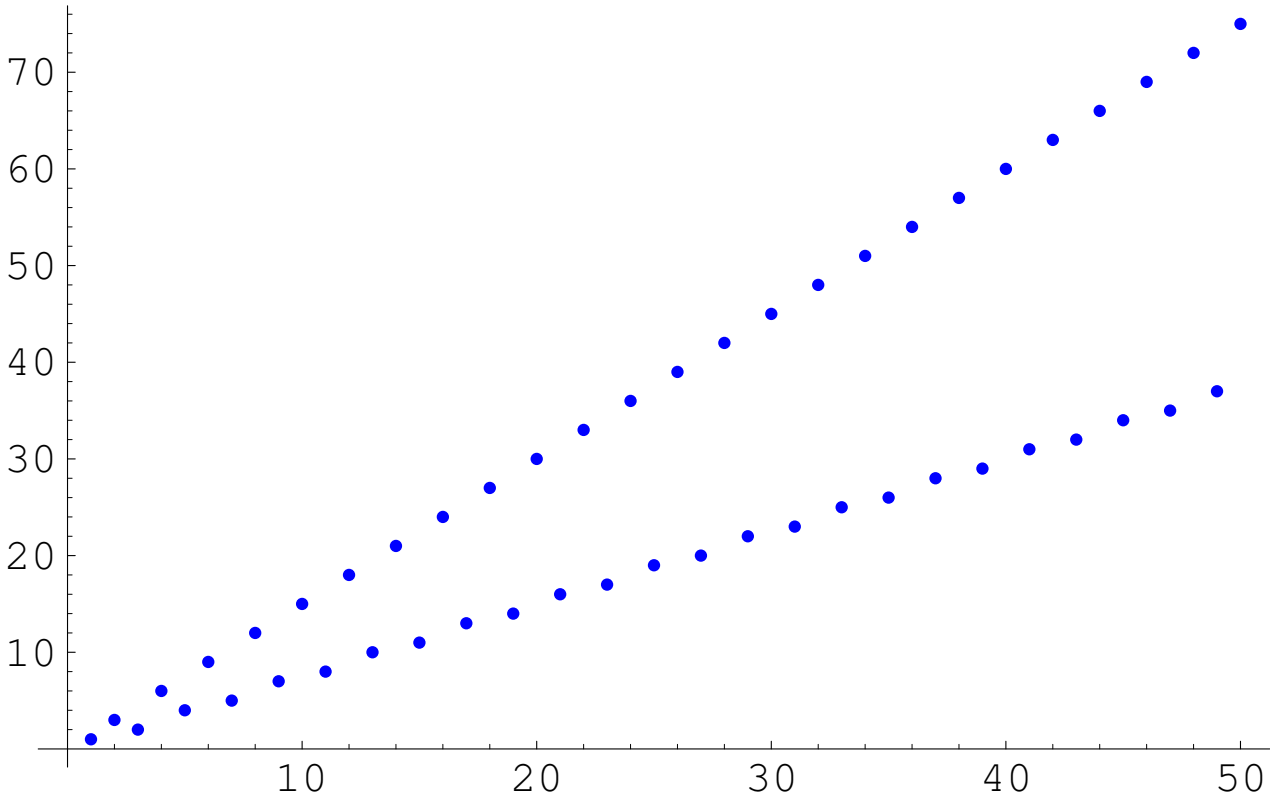
This is just $\mathrm{Cn}(6,3,6,3;0,3,1,2)$.

Proposition

$T : \mathbb{N} \to \mathbb{N}$ *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.

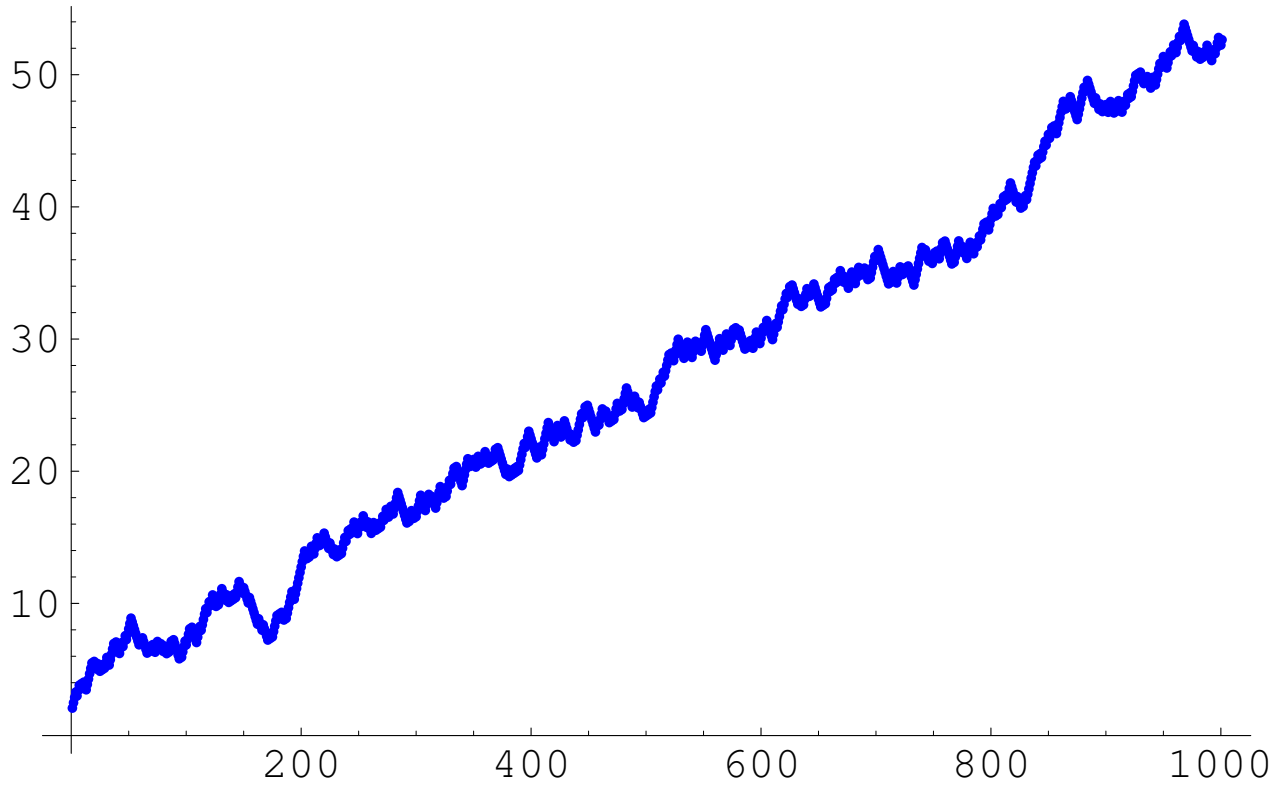
Known finite cycles are:

$$(1),$$
$$(2,3),$$
$$(4,6,9,7,5),$$
$$(44,66,99,74,111,83,62,93,70,105,79,59).$$

Open Problem:

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

Orbit of 8



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.

There is another way one can 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 has a positive solution.

Classical example: the continuum hypothesis or the axiom of choice in set theory.