Unsolvable problems in mathematics

Greg Hjorth

greg.hjorth@gmail.com

University of Melbourne

Faculty of Sciences
University of Adelaide
July 3, 2009

§0. Bordering on the obvious

It goes without saying that every mathematical problem has a solution. Maybe there might be technical difficulties in multiplying

$$2,300,244,472,363.01123$$

(i.e. some really large number) by

$$113,456,646,543,245,764,342.3436$$

(i.e. some even larger number), but in principle, with enough care and effort we could compute the answer.

In some cases there might be technical issues involving the fact that we cannot write down a precise solution in a finite number of symbols. For instance

$$\sqrt{34} = 5.830951895...$$

(the decimal expansion extends for infinitely many digits), but there is an exact solution which up to any desired degree of accuracy can be calculated.

There are more subtle versions of the same issue, such as the "three body problem." But those are technical issues based on our ability on our ability as finite, imperfect beings to calculate or represent solutions.

Fundamentally, though, there will be a solution or answer to every mathematical question.

We may or may not be able to calculate the right answer, but there *will be* a right answer.

In some sense this is so obvious as to not merit explicit statement, or even conscious recognition.

It is just obvious.

And also completely false.

In this talk I want to describe some mathematical problems which even *in principle* do not have a a solution.

I will describe when they were discovered.

Indicate why the existence of unsolvable problems, far from being surprising, was inevitable.

And go on to suggest that in some sense this phenomena of *unsolvability* is desirable.

§1. Back in the days of innocence

In the Paris conference of the *International Congress of Mathematicians* in 1900, David Hilbert proposed a list of 23 seminal mathematical problems.

The first of these referred to *Cantor's continuum hypothesis*.

In simplified language, this relates to the concept of *cardinality* or *size* for infinite objects.

There is a precise mathematical notion of *counting the size* of an infinite set such as the *natural numbers*,

$$\mathbb{N} = \{1, 2, 3, 4, ...\},$$

the *integers*,

$$\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\},$$

or the *real numbers*,

$$\mathbb{R} = \{x : x \text{ has an infinite decimal expansion}\}.$$

It turns out that $\mathbb{N}$ and $\mathbb{Z}$ have the same *cardinality*, but $\mathbb{R}$ has strictly larger cardinality.

Hilbert's first question begins with:

> *The investigations of Cantor on sets of points* [i.e. real numbers] *suggest a very plausible theorem, which nevertheless, in spite of the most strenuous efforts, no one has succeeded in proving.*

And from there he goes on to describe Cantor's hypothesis that $\mathbb{R}$ has the smallest possible infinite cardinality above that of $\mathbb{N}$.

In 1963, following earlier work of Gödel, Cohen established that Cantor's hypothesis is *formally undecidable*. If you like, *intrinsically unsolvable*.

In rough and somewhat simplistic terms this amounted to *proving* that no valid mathematical argument can be given to establish Cantor's hypothesis as true and yet equally no mathematical argument can be given to demonstrate it false.

In some ways this is the most celebrated example of a formally unsolvable mathematical problem. However there more important issue is not this celebrated question turned out to be undecidable.

The important issue is that undecidable problems are ubiquitous.

§2 The tenth Hilbert problem

In this problem he asks for a

> *process according to which it can be determined by a finite number of operations whether* [an equation with integer coefficients] *is solvable in rational integers,*

where here a *rational number* refers to a fraction of the form

$$\frac{n}{m}$$

for n, m in $\mathbb{Z} = \{..., -2, -1., 0, 1, 2, ...\}$.

Here are two quick examples of such equations.

$$x^2 = 2.$$

There are exactly two solutions.

$$x = \sqrt{2}$$

and

$$x = -\sqrt{2}.$$

It has been known from long before the time of Christ that neither of these are rational numbers, and hence here no such solution is possible.

On the other hand

$$x^2 + y^2 = 25$$

admits the solution $x = 3$ and $y = 4$.

Of course the phrase *finite number of operations* is necessarily someone vague. The customary modern reading is that Hilbert was implicitly asking for an *algorithm*, of the kind which could be programmed into a computer, for determining whether an equation of the indicated type has a solution over the rational numbers.

Here Matiyasevich showed in 1970 that, contrary to Hilbert's implicit expectation, no such algorithm is possible.

In essence his negative solution was based on what is known in theoretical computer science as the *unsolvability of the halting problem*.

§3 The halting problem

Sad but true. Sometimes our computers simply crash. They sit there endlessly churning their wheels, never coming to a conclusion, never *halting* some endless process.

How long should one wait? As a practical matter most of us are simply going to unplug and reboot when it is sitting idle for more than 5 or 10 minutes. Theoretically, though, there is no upper bound on how long it might take for the computer to find a way to complete its internal tasks and come back to life.

Thus one might fantasize about having a kind of meta-program, or *guardian* program attached to each copy of Windows or Linux which would be able to make predictions regarding when a computer program will *eventually* halt with some output – as against endlessly spinning its wheels.

Definition A *Turing machine* is a machine with finitely many states and operations equipped with an infinitely long *memory tape* and an infinitely long *input tape* and an infinitely long *output tape*.

On the memory tape it is able to write and read two symbols, "0" and "1".

It is only able to read the symbols on the input tape, but we for our part are able to write down any finite string of 0's and 1's and present those to the Turing machine as the input on some computation.

Finally, the machine is able to write on the output tape, which we take as the output for our requested calculation.

The definition just above is necessarily somewhat vague – at least for the time of this presentation.

However each part can, with sufficient time and effort, be made precise, and the critical aspect of this definition is that the Turing machine itself is a finite object.

Thus in principle it could be represented or encoded by a string of finitely many symbols.

Indeed if we are sufficiently careful, it can be represented with a finite string of 0's and 1's.

Theorem There is no Turing machine which can determine which Turing machines will eventually halt

In order to give the proof of this theorem, we would need to spend time making the concepts much more precise, but it is possible to give a rough idea of proof.

Keep in mind that every Turing machine can be encoded by some finite sequence $\vec{s}$ of 0's and 1's. Thus we can write $M_{\vec{s}}$ for the Turing machine corresponding to the finite binary sequence $\vec{s}$.

Suppose we had some *master* Turing machine, M, which given a description of a Turing machine $\vec{s}$ and a possible input $\vec{t}$ will output 0 if the computer $M_{\vec{s}}$ corresponding to $\vec{s}$ will halt on input $\vec{t}$ and output 1 otherwise.

Then from M we could form a new Turing machine N which on input $\vec{s}$ halts with value 0 if the machine $M_{\vec{s}}$ *presented with* $\vec{s}$ halts with value 1 and otherwise outputs 1. The key thing here is that N will halt on *every input*.

But the problem is that N will itself be equal to some such $M_{\vec{s}}$. Yet N's output on this $\vec{s}$ will be different to the output of $M_{\vec{s}}$, with a contradiction.

§4 The Gödel incompleteness theorems

There was a certain view point which comes across in some of Hilbert's philosophical writings. To some extent it is implicit in both the tenth Hilbert problem, mentioned above, and the second Hilbert problem on the *consistency* of arithmetic.

It is a view which was ultimately shattered by the Gödel incompleteness theorems.

At the cost of somewhat simplifying his position, here is a brief summary in four principles. The fourth of these roughly corresponds to the *second* Hilbert problem.

1. The methodology of mathematics can be enumerated in finitely many principles of reasoning.
2. A correct mathematical proof consists of a sequence of statements, each of which can be mechanically verified to follow from the earlier statements and our finite list of principles.
3. Those principles never lead to a contradiction or any obviously absurd statement.
4. Every mathematical statement involving finite objects can be either proved or disproved using our finitely many principles.

That deceptively seductive manifesto was exactly the target of Göel's first incompleteness theorem.

This can actually be seen from unsolvability of Halting problem.

Although in this talk I have been rather loose with the definitions, those four principles can be made mathematically precise, and Gödel's achievement was to show that not only did the formal system of mathematical reasoning Hilbert had in mind fail to satisfy those principles, but there is *no* formal system which does.

The point is that a formal mathematical proof is a finite object and a Turing machine could be designed to verify whether a finite object actually corresponds to a correct formal proof. Thus if one also accepts that every mathematical statement of the appropriate kind is either provable or refutable, then we could design a Turing machine which when presented with a mathematical statement concerning finite objects could determine whether the statement was provable or refutable, and hence true or false.

For instance, given some burning mathematical question, such as "Is the twin prime conjecture true?" the Turing machine would successively write down all lists of finitely many mathematical statements, verify in turn whether they were a proof, and stop when one of them either proves or refutes the conjecture.

In this way we would have a kind of *Master Mathematician* Turing machine. Any question could be answered, simply by waiting until the machine finally returns an answer after examining all possible proofs.

However the statement that the Turing machine $M_{\vec{s}}$ halts on input $\vec{t}$ is itself a statement about finite objects.

Thus our Master Mathematician would also be able to determine in a finite number of time whether the Turing machine $M_{\vec{s}}$ halts on input $\vec{t}$.

Which is exactly to say that it could solve the halting problem – and the halting problem is not solvable in that sense.

§5 Effectiveness of solutions

Previously we discussed the idea of *Turing machine* as a kind of mathematical idealization or explication of the concept of computer. The notion of what is computable then becomes an issue of whether we can design a Turing machine which is certain to *eventually* halt with the correct answer as its output.

As a practical matter, this is less than enthralling. We would probably look askance at a computer which took a thousand years to complete the simplest task. We want something to be not just computable, but *effectively* computable.

There is a generally accept notion of what it means to be *effectively computable*

Definition Let $\{0,1\}^{<\infty}$ denote the collection of all finite binary strings. For $\vec{s}$ in $\{0,1\}^{\infty}$ we let $\mathrm{lh}(\vec{s})$ denote its length.

For instance

$$\mathrm{lh}(010001) = 6$$

while

$$\mathrm{lh}(110) = 3.$$

Definition A function

$$f : \{0,1\}^{<\infty} \to \{0,1\}^{<\infty}$$

is said to be *in P* or *polynomial time computable* if there are constants

$$k, n$$

in $\mathbb{N}$ and a Turing machine M such that for every $\vec{s}$ in $\{0,1\}^{<\infty}$ we have that M halts with output

$$f(\vec{s})$$

in fewer than

$$k \cdot (\mathrm{lh}(\vec{s}))^n$$

seconds.

This definition of polynomial time is a rather subtle way of drawing out the idea of *effectively computable*. The polynomial bound could be very poor, and it might be far in excess of the time we are willing to wait.

In defense of the definition, to start analyzing the concepts *mathematically* we probably want a notion which is not dependent on how long it takes *currently* with existing hardware to calculate a function. The definition of polynomial time is essentially hardware independent.

This in turn suggest a kind of modern day question, asking for an analogue of the Gödel incompleteness theorems.

A potentially broader class of functions than those in P are those which are computable and whose output can be *verified* in P. That is to say, those functions

$$f : \{0,1\}^{<\infty} \to \{0,1\}^{<\infty}$$

for which:

1. There is a Turing machine M which can compute $f(\vec{s})$;
2. and there is another Turing machine N which given $\vec{s}$ and $\vec{t}$ can compute in polynomial time in $\mathrm{lh}(\vec{s})$ whether $f(\vec{s})$ equals $\vec{t}$.

Here one is led to the question of whether every function of that form is in fact in P. That is to say, does this concept reduce to the previous notion. This is basically the notorious open question of whether P equals NP.

In some very loose sense, $P \neq NP$ would represent a modern version of the incompleteness phenomena uncovered by Gödel and the unsolvability of halting problem.

Note there is an asymmetry running through all the examples considered so far.

Once we have a possible mathematical proof of say the twin prime conjecture, the verification that it is actually a proof should be routine and mechanical. The Gödel incompleteness theorems assert, among other things, that there is no mechanical procedure for determining in advance *existence* of a proof.

Once we have a proposed integer solution to some sequence of equations, it is mechanical to verify whether it is in fact an actual solution. Matiyasevich's negative solution to the Hilbert problem asserts that there is no mechanical procedure which in advance can find a solution.

Once a Turing machine halts in a computation we can be happy with the outcome, but the unsolvability of halting problem states that there is no way we can mechanically determine in advance whether the machine will in fact eventually halt.

If $P \neq NP$ then there is a class of problems where once we have a proposed solution we can *effectively* verify whether it is an actual solution, but in advance there is no way to effectively find a solution.

So much for the analogy, now for the crucial difference.

Once upon a time it might have been nice to have been able to solve the halting problem. Maybe Hilbert would have been overjoyed with a positive solution to his tenth problem. Maybe many mathematicians were disappointed in Gödel's demolition of the Hilbert program.

But, right now, as the world presently stands, it would be an absolute disaster of the highest order if P were to turn out to be equal to NP.

§6 Public key encryption

All of internet commerce and much of the world's military uses *public key encryption*, which in some sense tries to exploit a supposed gap between P and NP.

In the specific form of internet banking, it can be summarized in the following way.

1. My computer wants to send information down the internet which encodes my credit card to a purveyor which sells goods through a web site.
2. Any information that passes between us goes through the internet and can potentially be accessed by other computers on the web.
3. Thus the purveyor provides some information, a *public key*, which I can use to encrypt my credit card.

4. This is done in such a manner as:

   (a) My computer can encrypt the number *effectively*;

   (b) the company at the other end can effectively, which is to say in P time, decode the communication I have sent them based on knowing the solution to some NP problem;

   (c) an eavesdropper on the world wide web might be able to access the public key and my message back to the purveyor, but will not be able to decode my card number in polynomial time.

The most commonly used form of public key encryption is RSA encryption. This roughly works as follows:

1. The purveyor chooses some sequence of prime numbers, $p_1, p_2, ..., p_n$.
2. They then multiply them together to get $k = p_1 \cdot p_2 ... \cdot p_n$. This can be done in polynomial time. (That is to say, polynomial number of seconds in the lengths of the binary representations of the indicated primes.)
3. Then they transmit k to me through the internet – where it can be potentially read by an eavesdropper.
4. I use k to encode the credit card number and transmit it publicly back to the purveyor.
5. Using $p_1, ...p_n$ the purveyor can decode the credit card number in polynomial time.

However the whole security of the process rests on the reconstruction of $p_1, ...p_n$, the prime factors from k being in P but not NP.

§7 Summary

The incompleteness phenomena was unpleasant, even shocking, when uncovered in the first part of last century.

From the perspective of modern day mathematicians, it appears absolutely inevitable.

Its modern day variant or analogue would be the statement

$$P \neq NP,$$

which not only now seems plausible, but highly desirable.