Hilbert's Tenth Problem

Marcus Collins

July 2026

Diophantine Equations

Hilbert's tenth problem is regarding Diophantine Equations.

Definition

A *Diophantine Equation* is a polynomial (possibly in multiple variables) with integer coefficients that allows only integer solutions.

Figure: Diophantus of Alexandria

Examples

Diophantine Equations

$$3x - 5 = 0$$

$$6xy + 5x^2y - 13y^3 = 0$$

$$x^3 + y^3 - z^3 = 0$$

Not Diophantine Equations

$$x^y - 8 = 0$$

$$\ln(x) - 1 = 0$$

$$\sqrt{x} + \sqrt[3]{y} - 16 = 0$$

Hilbert's Tenth Problem

At the turn of the 20th century, David Hilbert enumerated 23 unsolved problems. The 10th on this list was to find an algorithm to determine if a given Diophantine equation has solutions. The negative answer was finished in 1970

Theorem (Matiyasevich, Robinson, Davis, Putnam)

There is no algorithm to determine whether a given Diophantine Equation has solutions!

Figure: David Hilbert

Fun with Fruits

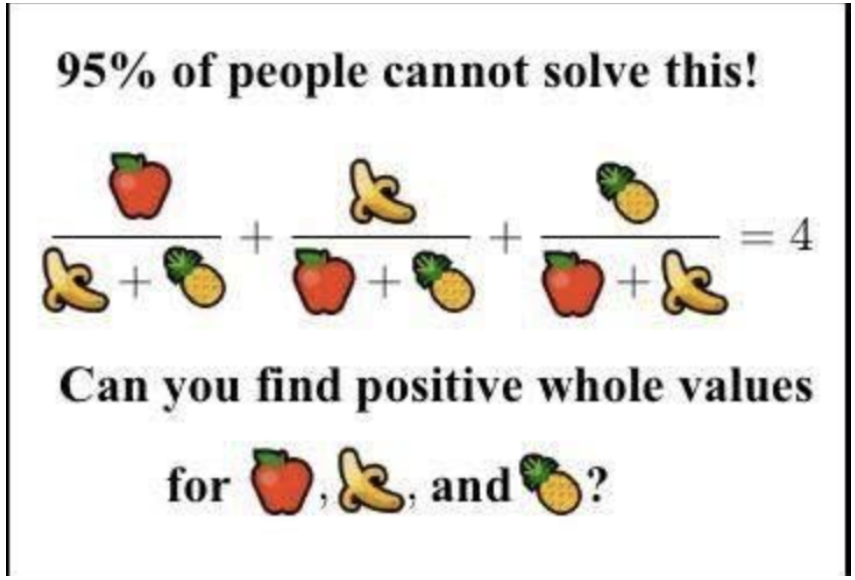

Figure: The smallest solution is apple = 154476802108746166441951315019919837485664325669565431700026634898253202035277999, banana = 368751317941299998271978115652254748254929799689719709962831374716372246340556 79, pineapple = 437361267792869725786125260237139015281653755816161361862143799337842346777 2036

Some Helpful Simplifications

We can reduce a system of equations to a single equation with the following:

Lemma

A system of equations $P_1(x_1, \ldots x_m) = 0, P_2(x_1, \ldots x_m) = 0, \ldots P_n(x_1, \ldots x_m) = 0$ *has a solution if and only if* $P_1(x_1, \ldots x_m)^2 + P_2(x_1, \ldots x_m)^2 + \cdots + P_n(x_1, \ldots x_m)^2 = 0$ *has a solution*

We can use this, alongside a theorem of Lagrange, to reduce Hilbert's tenth question to only positive integers:

Lemma

If there is no algorithm for finding positive solutions to Diophantine equations, there is no algorithm for finding integer solutions to Diophantine equations.

Some Helpful Simplifications Continued

Lemma

If there is no algorithm for finding positive solutions to Diophantine equations, there is no algorithm for finding integer solutions to Diophantine equations.

Proof.

An algorithm that could find integer solutions would be able to find positive solutions to $P(x_1, \ldots x_m)$ by adding the additional equations
$x_1 = p_1^2 + q_1^2 + r_1^2 + s_1^2 + 1, \ldots x_m = p_m^2 + q_m^2 + r_m^2 + s_m^2 + 1$ using the previous lemma. □

Decidable, Semi-decidable, and Diophantine Sets

Definition

A set S is called *decidable* if there exists an algorithm such that for any given a, the algorithm can determine in a finite amount of time if $a \in S$.

Definition

A set S is called *semi-decidable* if there exists an algorithm that will verify for a given element a if $a \in S$ after a finite amount of time, but may run forever if $a \notin S$.

Definition

A set S is called *Diophantine* if there exists a polynomial $P(a, x_1, x_2, \ldots x_m)$ such that
$a \in S \iff \exists(x_1, \ldots, x_m) P(a, x_1, \ldots x_m) = 0.$

It is clear that every decidable set is semi-decidable and every Diophantine set is semi-decidable.

Proof Outline

If Hilbert's tenth problem has an algorithm, Diophantine sets will be decidable. The proof shows that this is not the case by first showing there exists semi-decidable sets that are not decidable (with the Halting problem), and then by showing the shocking fact that every semi-decidable set is Diophantine.

Theorem (Turing)

There exists a semi-decidable set that is not decidable.

Up to this point, we have avoided discussing what we mean by "algorithm." A formal treatment is beyond the scope of this talk, but we will use the notion of a Turing machine.

An Informal Treatment of Turing Machines

We will accept that all reasonable algorithms can be modeled by a Turing machine. We assume the following:

Assumptions

1. There are a countable number of algorithms, and thus an algorithm can be expressed just as an integer
2. Every algorithm will take in an integer as an input, then either accept it, deny it, or run forever

For our purposes, we will take anything we can describe in plain language as a valid algorithm. For instance, to check if n is prime we could say "For all integers x such that $1 < x < n$", check if $x|n$. If any x divides n, return composite. Otherwise, return prime."

The Halting Problem

Theorem

There is no algorithm that can determine whether any given algorithm will halt on any given input.

Proof.

Suppose that such an algorithm existed. Define a second algorithm as follows: Given any algorithm, run it on itself. If it halts, have our algorithm enter an infinite loop, if it does not halt, have our algorithm halt. What happens when we run this algorithm on itself? If it halts, that means running it on itself should have entered into an infinite loop. If it enters into an infinite loop, that means running it on itself should have halted. Thus this is all impossible, and no algorithm to determine halting can exist. □

From this, we can say that the "Halting set" (the set of all inputs on which a Turing machine halts) is semi-decidable but not decidable.

Every Semi-Decidable set is Diophantine: An Outline

Theorem

For a semi-decidable set S,
$x \in S \iff \exists y \forall_{k \leq y} \exists z_1 \ldots \exists z_m P(x, y, k, z_1, \ldots z_m) = 0$ for some polynomial P with integer coefficients.

Theorem

For a semi-decidable set S,
$x \in S \iff \exists z_1, \ldots z_m E(x, z_1, \ldots z_m) = 0$, where E is an exponential Diophantine equation.

Theorem (Matiyasevich, 1970)

There is a polynomial $P(a, b, c, x_1 \ldots x_n)$ such that
$a^b = c \iff \exists x_1, \ldots x_n P(a, b, c, x_1, \ldots x_n) = 0$.

Thanks!

Figure: Some ducks