Number Theory Meets Computability Theory

Kyle Richardson

May 2020

Asking Questions about Equations In this article, we consider the problem of solving certain types of equations (called *polynomial equations*). For example, imagine you are given the equation

$$4 - x^2 = 0$$

and asked to do the following: determine if there exists a solution for the variable x in integers. Since it is easy to come up with a single solution in this case, namely $x = 2$, we can use this specific solution to answer this question in the affirmative. In contrast, it should be clear that a slightly modified equation such as $2 - x^2 = 0$ *does not have a solution*, which can easily be checked by hand.

Of course, not all equations of this form are as straightforward to solve. If we allow for a few additional variables and slightly larger constants, we quickly stumble upon innocent looking equations such as the following[1]:

$$(x^3 + y^3 + z^3) = 114,$$

whose solution (if it exists) continues to elude the many number theorists who are actively working on this and other related equations involving *sums of three cubes* of the form $x^3 + y^3 + z^3 = a$. As before, it suffices to find a single solution for variables x, y, z, however searching the infinite space of integers, especially in the absence of a broader mathematical theory, can easily lead one astray[2].

We can also ask seemingly more complicated follow-up questions, such as whether the specific equation below (known as the *Ljundgrenn* equation):

$$x^2 - 2y^4 + 1 = 0 \tag{1}$$

[1]This example is taken from Poonen (2008). Other examples and explanations are adapted throughout from the following very readable surveys: Smith (2011); Pasten (2019)

[2]Considerable empirical progress was made in 2019 on sum of three cubes problems when solutions for $a = 33$ and $a = 42$ were discovered by Andrew Booker and colleagues (see Booker (2019)). In the former case, his investigation involved looking at positive and negative integers in the range of 10^{16}, which required the equivalent of 23 years of continuous computation on a single computer; this resulted in the following highly unintuitive variable solutions: $(x = 8866128975287528, y = -8778405442862239, z = -2736111468807040)$. In the latter case, finding a solution required (the equivalent of) 1.3 million hours of compute time, which is likewise an unfathomable amount of computation time.

not only has a solution, but a fixed set of unique solutions (the Norwegian mathematician Wilhelm Ljundgreen, after whom the equation is named, proved in 1942 that this equation indeed only has the following two solutions: $(x = 1, y = 1)$ and $(x = 239, y = 13)$). We can also ask whether *no solutions exist* for a given equation under certain conditions, for example whether it is true that the following equation (or family of equations):

$$x^n + y^n = z^n, \tag{2}$$

has *no integer solutions* for any $n > 2$, as was first conjectured by Pierre Fermat in the 17th century and proved by Andrew Wiles in 1994[3]. Answering questions of this type, which are commonly found in number theory proper, often require considerable amounts of mathematical sophistication and ingenuity.

Rather than focusing on solving specific equations and coming up with specialized solutions, as number theorists tend to do, the main focus of this article will be on a larger and much more grandiose question, namely: does there exist a universal solution for solving arbitrary (polynomial) equations? In other words, can we automate the process of equation solving and devise a *universal algorithm* that can determine, given any polynomial equation regardless of its number of variables or its difficulty, whether it has a solution?

The Search for a Universal Equation Solver A version of this last question was asked by the German mathematician David Hilbert in 1900, and is problem number 10 of 23 in the famous **Hilbert Problems** (we examine the exact phrasing of his question in the next section). In the words of Martin Davis, such problems were among the premier mathematical problems *that the nineteenth century [had] left for the twentieth century to solve.* Given the long history of work on polynomial equation solving, dating back to Diophantus in the 3rd century AD and even before, the idea of a universal algorithm is not only grandiose but, as Davis once wrote, utopian.

The 20th century did solve this problem, though the solution likely would have baffled and annoyed David Hilbert, who had dreamed of reducing all of higher mathematics to a definitive set of formal axioms and algorithmic principles. In 1970, the Russian mathematician Yuri Matiyasevich, building on the work of a rather eclectic and tenacious group of American researchers that includes Martin Davis, Julia Robinson and Hilary Putnam, provided the final piece in the proof that ultimately lead to the following *negative solution*: no such universal method or algorithm exists for solving arbitrary polynomial equations (or more specifically, what are called *diophantine equations*). A large part of the final proof involves a rather astonishing and unexpected link between number theory and basic concepts from computer science and *computability theory.*

While this result closely relates to other *impossibility* results discovered in the early days of computer science by Kurt Gödel, Alonzo Church, Alan Turing, Emil Post[4] and others, it is not at all obvious at first sight that Hilbert's 10th problem is a problem that computer science would

[3]The story behind this conjecture is likely to be the most repeated anecdote in mathematics. Fermat had apparently scribbled this conjecture in 1637 into the margins of Diophantus' *Arithmetica* and claimed that he had *discovered a truly remarkable proof* that was too complex to fit in the margins. As mentioned above, the ultimate proof didn't arrive until over 350 years later.

[4]Emil Post famously had the following to say about Hilbert's 10th problem nearly 25 years before its final resolution: it 'begs for an unsolvability proof'.

have much to say about. One might have thought that any algorithmic solution to the general problem of solving (polynomial) equations would rely solely on principles from number theory and arithmetic. This turns out not to be the case and one of the big technical ideas to come out of this work is that one can link notions from computer science about *computable* or *recursive* sets with sets of solutions to equations.

To see how this works, consider the general form of the three cube problem shown below (which we might think of as denoting the *family* of all the three cube equations):

$$x^3 + y^3 + z^3 = a. \tag{3}$$

Solutions to this equation (i.e., particular values a for which there exist solutions for the variables) can then be represented as sets of the following form:

$$S = \Big\{ a \mid x^3 + y^3 + z^3 = a \textit{ holds for some integers } x, y, z \Big\},$$

Asking whether Equation 3 has solutions for a particular a (e.g., 114) in this setting then reduces to asking whether a is in the set S (e.g., $114 \in S$?). Similarly, asking whether there are a unique set of solutions, or whether no solutions exist, involves asking questions about the size and scope of S (is S the empty set? is S of size k?,...). When formulated in this way, the problem quickly starts to look like the types of problems encountered in theoretical computer science.

At its heart, Hilbert's 10th problem is what we now call a *decision problem*; what we want to prove is that there exists an algorithm that can decide (i.e., return a yes/no answer) whether a given equation has a solution or not, or equivalently whether a number is included in a set of solutions. Based on the work of Alonzo Church and Alan Turing, who were among the first to study modern decision problems, we know that some problems can be *undecidable*; that is, it can be proven that no such algorithm can exist no matter how hard one tries. This is what happens in the case of Hilbert's 10th problem: it is possible to define hypothetical sets of solutions for which it is provably impossible to build an algorithm that can decide set-membership. As a consequence, this undecidability proves that a general algorithm cannot exist (the surprising part is that we can talk about such sets without having to say very much about their corresponding equations, and in a way that entirely side-steps the practical issues involved with *explicitly* finding its members).

Using the language of computability theory, we can specifically say the following: any *positive* solution to Hilbert's 10th problem (i.e., proof that a universal algorithm exists for solving arbitrary diophantine equations) would imply a positive solution to the Halting Problem of Turing (1936), which is the most well-known undecidable problem in computer science. The goal of this article is explain what this means and to provide enough of the technical computer science background that is needed to sketch out this fascinating result.[5]

Diophantine Equations and Sets

First, let's consider Hilbert's original description of the problem:

[5]We will only give a cursory overview of the number theoretic aspects of this problem that helped Matiyasevich and others to arrive at the final solution. The full details of this can be found in the surveys Davis (1973) and Jones and Matijasevič (1991), as well as Matiyasevich (1993).

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 in a finite number of operations whether the equation is solvable in rational integers.

We will consider each part of this problem statement in turn. Hilbert's notion of a *process ..[involving] a finite number of operations* is what we would now call an *algorithm*, which was a rather fuzzy concept in 1900 (we will talk about algorithms in the next section). By *diophantine equation*, Hilbert's is referring to certain types of polynomial equations that characterize most of the equations we have considered so far. A *polynomial* in our case will mean the following:

Definition 1. A **polynomial** expression over n variables/*unknown quantities* $x_1, x_2, .., x_n$, denoted as $p(x_1, ..., x_n)$, is any finite sum of *monomials*, or expressions of the form:

$$cx_1^{k_1}, ..., x_n^{k_n},$$

where c is an integer *coefficient* (i.e., positive and negative integers and zero, denoted as $\mathbb{Z}$) and $k_1, ..., k_n$ are natural numbers including zero (denoted as $\mathbb{N}$) that are, importantly, distinct from the polynomial's variables.

Examples polynomial expressions include $x_1^2 - 4x_1 + 3$ (where, for convenience, subtraction is used in place of addition with a negative number, $+ - 4x$), $4x_1^3 + 6x_2$, $x_1 + x_2 + ... + x_4$ (with all coefficients c equal to 1) and so on.[6] When talking about polynomials it is important to specify the range of their variables. Diophantine equations are special types of polynomial equations that restrict the range of variables in the manner specified below.

Definition 2. A **diophantine equation** is specific type of polynomial expression $p(x_1, ..., x_n) = 0$ (also known as a *polynomial equation* in *traditional form*) restricted to *integer* unknowns $x_1, ..., x_n$ (or what Hilbert calls *rational integers*).

We will show momentarily that it suffices to modify the problem such that variables are restricted to natural numbers, which is a inconsequential variant of Hilbert's original description. We also note that it is sometimes easier to transform diophantine equations out of their *traditional form* $p(\cdot) = 0$ into equations of the following type:

$$p_l(x_1, ..., x_n) = p_r(x_1, ..., x_n) \tag{4}$$

where p_l and p_r are two separate diophantine equations defined over the same variables. For example, transformations of this type become convenient when we want to remove negative terms in an equation, which we might do with the following non-trivial diophantine equation:

$$4x^3y - 2x^3z^3 - 3y^2x + 5z = 0,$$

to arrive at:

$$4x^3y + 5z = 2x^2z^3 + 3y^2x$$

by transposing the negative terms.

[6]When trying to map specific polynomials into a sum of monomials in the form provided, it is important to recall that each k_j exponent can be 0, which maps any number to 1. Therefore, in $p(x_1, x_2) = 4x_1^3 + 6x_2$, the first *term* $4x_1^3$ in the sum (whose coefficient is 4) is equal to $4x_1^3x_2^0$, whereas the second term is equal to $6x_1^0x_2^1$. Likewise, for any term without an explicit coefficient, it can be assumed that the coefficient is 1.

As briefly discussed in the last section, one of the big ideas is to define diophantine equations with additional variables called *parameters* a, as in $p(a, x_1, x_2, ..., x_n)$ (where $x_1, ..., x_n$ continue to be what we previously called *unknowns*), which allow us to describe abstract *families of diophantine equations*. This gives rise to an important concept called a *diophantine set*.

Definition 3. A **diophantine representation** of a diophantine equation p with integer *unknowns* $x_1, ..., x_n$ and a *parameter* a[7] is the set (which we will henceforth call a **diophantine set**):

$$S = \Big\{a \mid \exists x_1, ..., x_n[p(a, x_1, ..., x_n) = 0]\Big\} \tag{5}$$

We will also say that a given set of numbers $\{a_1, a_2, ...\}$ *is diophantine* if and only if it has a diophantine representation.

Interestingly, this set construction will allow us to talk about number theoretic concepts without having to say very much about specific equations. That is, unlike in ordinary number theory where one usually starts with a specific equation or family of equations and attempts to arrive at a set of solutions and parameters, the idea here is that we will start with a set and try to arrive at a family of equations to demonstrate that the set is diophantine.

Examples Let's illustrate this idea by starting with the set of natural numbers $\mathbb{N}$. Is this set diophantine? The answer is yes, which can be demonstrated by exploiting the following important theorem:

Theorem 1. *Four-square theorem (Lagrange 1770) Any natural number a can be expressed a sum of four integer squares:*

$$a = x_0^2 + x_1^2 + x_2^2 + x_3^2 \tag{6}$$

Using this equation, we therefore have our polynomial equation that defines exactly the natural numbers. To make it have the desired diophantine form, we can simply move a to the other side of the equation to arrive at $a - (x_0^2 + x_1^2 + x_2^2 + x_3^2) = 0$. Now to ask if a given number such as 4 is in the set of natural numbers, we can also also ask whether $4 - x_0^2 + x_1^2 + x_2^2 + x_3^2 = 0$ has a solution.

What about the set of *composite numbers* (i.e., non-prime positive integers): $\{4, 6, 8, 9, 10, 12, ...\}$? Here we have a slightly less intuitive equation:

$$(x_1 + 2)(x_2 + 2) - a = 0,$$

which again gives us what we need to say that composite numbers are diophantine. How about the set of composite numbers that are odd? Well, we already have an equation for composite numbers, and we can express odd numbers with the following equation:

$$2x_1 + 1 - a = 0$$

[7]We note that it is also possible to consider equations with tuples of parameters, $(a_1, .., a_m)$, however our simplified version will suffice to prove the main result.

Defining this set can then be accomplished by combing both equations and squaring each individual equation (i.e., to ensure that they evaluate to 0):

$$(2x_1 + 1 - a)^2 + ((x_2 + 2)(x_3 + 2) - a)^2 = 0$$

In general, rather than talking about individual equations or family of equations, we can even discuss systems of equations by combing multiple diophantine equations (or decomposing complex equations into simpler ones) in the manner illustrated above.

Solutions in natural numbers The four-square theorem introduced above can be used to make a few important points about diophantine equations as detailed in (Matiyasevich, 1993, Chapter1). First, if you imagine that we have the following diophantine equation:

$$p(x_1, .., x_n) = 0, \tag{7}$$

with solutions in arbitrary integers $\mathbb{Z}$, then clearly this solution includes solutions in natural numbers since $\mathbb{N} \subset \mathbb{Z}$. On the other hand, if this equation has a solution in natural numbers, then this solution also includes a solution in arbitrary integers since we can use the four square theorem (Equation 6) to rewrite each x_j in the equation as:

$$\begin{aligned} x_1 &= y_{1,1}^2 + y_{1,2}^2 + y_{1,3}^2 + y_{1,4}^2 \\ x_2 &= y_{2,1}^2 + y_{2,2}^2 + y_{2,3}^2 + y_{2,4}^2 \\ &\dots \\ x_n &= y_{n,1}^2 + y_{n,2}^2 + y_{n,3}^2 + y_{n,4}^2 \end{aligned}$$

resulting in:

$$p((y_{0,1}^2 + y_{0,2}^2 + y_{0,3}^2 + y_{0,4}^2), ..., (y_{n,1}^2 + y_{n,2}^2 + y_{n,3}^2 + y_{n,4}^2)) = 0,$$

where each $y_j \in \mathbb{Z}$. The important point is the following: *To establish the unsolvability of Hilbert's 10th problem in its original form, it is sufficient to establish the unsolvability of its analog for non-negative solutions* (Matiyasevich, 1993). For this reason, we limit ourselves exclusively to variables in natural numbers. *Thus, the problem of solving Hilbert's problem will crucially rely on understanding properties of set of natural numbers* and algorithms over sets, which we turn to in the next section.

Computability Theory

In this section, we detail the basic ideas from computability theory that inform us about sets and their algorithmic properties. This centers around a discussion of *recursively enumerable* and *recursive* sets and, most importantly, understanding the differences between both classes of sets.

Definition 4. A **recursively enumerable** (or *listable,computable,semi-decidable*) set is any subset $A \subseteq \mathbb{N}$ for which there exists an algorithm/Turing Machine/program that can print, with possible repeats, all the members of A and nothing more[8].

[8]Note that we can make this definition more complex both by considering subsets of $\mathbb{N}^m$ (i.e., sets of $m-$tuples over $\mathbb{N}$), or sets over $\mathbb{Z}$, however our restricted definition will suffice for proving our main results.

We will not delve into what exactly an algorithm means in this context, other than to say that it can be any effective procedure that solves the task at hand. Evoking Church's thesis (Church, 1936), an algorithm being *effective* means that there should exist an accompanying Turing Machine, which one might think of as a (correct and precise) mathematical model or simulation of the target algorithm and its hardware. The following result connects these notions more directly with ordinary functions:

Lemma 1. *The following definitions are equivalent:*

1. *A is recursively enumerable (according to Definition 4).*
2. *A is empty or in the range of a total (computable) function* $f : \mathbb{N} \rightarrow \mathbb{N}$
3. *A is in the domain of a partial (computable) function* $f : \mathbb{N} \rightarrow \mathbb{N}$

We leave the proof an an exercise to the reader (though we describe an example below). We will also gloss over the meaning of a *computable function*, though it is worth pointing out that virtually all numerical functions encountered in ordinary mathematics (e.g., addition, multiplication, exponentiation, integer square root, ...) are computable[9].

Example Let's take the set of square numbers considered already, i.e., $A = \{1, 4, 9, 16, 25, ..\}$. Is this set recursively enumerable? Yes, according to the following argumentation. We know that the following (diophantine) equation only has solution for a only when a is a square number:

$$a - x^2 = 0,$$

which we can exploit to define the following (computable) function (defined for all $x \in \mathbb{N}$):

$$f(x) = x^2$$

Putting all this together, it is clear that the range of this function i.e.,

$$\text{RAN}(f) = \Big\{ f(1), f(2), f(3), f(4), .. \Big\}$$

will only be square numbers and that A is recursively enumerable (thus satisfying condition 2).

What about condition 3? Here we can define the square root function for $sqrt : \mathbb{N} \rightarrow \mathbb{N}$ (which is a computable function). Clearly, this function is a *partial function*, or its domain is only a subset of $\mathbb{N}$, which is exactly the set of square numbers.

Notice that we didn't need to mention anything about an explicit algorithm or Turing Machine to make this case. What's more, we did the following curious thing (which started in the last section): we started by asking a question about an explicit set (which is often the starting point for many computer science or logic problems), then transformed this set into an equation. We could

[9]More specifically, virtually all ordinary functions in math are *primitive recursive functions*, meaning that they can be derived from 3 primitive computable functions (the successor function, the constant function and projection function) coupled with three operations called *(primitive) recursion, minimization and composition.*

have gone the other way, i.e., start with an equation (which is often the starting point for a number theorist), then move from that equation to a set (then directly to an algorithm).

One alternative way of defining a recursively enumerable set, which is consistent with the definition provided, is a set where there exists an algorithm for verifying membership among inputs that are in the set. Generalizing from this, there is a more general class of sets that imposes stricter conditions:

Definition 5. A **recursive** (or *computable,decidable*) set is any subset $A \subseteq \mathbb{N}$ for which there exist an algorithm that can determine (full) *set membership* (i.e., definitively decide for any arbitrary number x whether $x \in A$ or $x \notin A$).

Using the following result (as before, we leave the proof as a exercise for the reader):

Lemma 2. *If a set A and its complement $\overline{A}$ are both recursively enumerable, then A is recursive.*

we can demonstrate that the set of square numbers is recursive by showing that the set of non-square numbers is recursively enumerable. Here we will sketch an *algorithm* for doing this, which does the following: loops through/enumerates each $i \in \mathbb{N}$, and print i in the case when $\text{SQRT}(i)$ (which is a computable function) does not return a whole number, and simply ignore the rest[10].

The most important result for Hilbert's 10th problem from computer science is that not all recursively enumerable sets are recursive, which we examine in some detail below.

Not all recursively enumerable sets are recursive One common way to discover recursively enumerable sets that are not recursive is by exploiting the undecidability of the *Halting Problem*, which is one of the most famous theoretical results in computer science that was proved by Alan Turing in Turing (1936). We take a brief detour to discuss this problem, then as a corollary provide an explicit set of numbers that it recursively enumerable though not recursive.

Definition 6. In our simplified version of the Halting Problem, we will consider the following set K that we call the **Halting Set**:

$$K = \Big\{ (M_x, y) \mid \text{program } x \ (M_x) \text{ halts on input } y \Leftrightarrow M_x(y) \downarrow \Big\}$$

which consists of all pairs of Turing machines M_x identified by the integer code x (importantly, we will assume that we can assign numerical ids to all Turing machines M, which is a technique often referred to as *Gödelization*) and inputs y such that M_x *halts* or terminates on input y, which we denote using the symbol $\downarrow$ (in contrast, $\uparrow$ will either mean *never halts* or *undefined*). The **Halting Problem** is therefore the problem of determining membership in K given any arbitrary M_x and y.

[10] Clearly this algorithm is impractical since it requires looping through an infinite number of numbers in $\mathbb{N}$. When we argue about the existence of algorithms in mathematics, we are allowed to make unrealistic, even outlandish, assumptions about the amount of resources and time we are allowed (e.g., that we have an infinite of time/memory/-parallel computations/..). Part of the reason why we describe abstract algorithms here in terms of Turing Machines, as opposed to Python or Java programs, say, is that they permit such excesses (e.g., by providing a infinite memory in the form of an *infinite memory tape* on which we can read and write).

This definition seem simple enough, however it leads to difficulties centering around cases where the indices x and inputs y match one another (i.e., the numeric id x happens to match the input y). These give rise to *diagonalization arguments* of the sort first discovered by Cantor in relation to infinity.

Theorem 2. *The Halting Problem is undecidable (i.e., there exists no universal algorithm/program/Turing machine for deciding membership in K).*

Proof. Let's imagine that K is decidable (hence making the Halting Problem decidable). Then it is possible to define another Turing machine M' that does the following (since K being decidable would allow us to compute the membership conditions on the right):

$$M'(x) = \begin{cases} 0 & \text{if } M_x \text{ does not halt on } x \Leftrightarrow (M_x, x) \notin K \\ \uparrow & \text{if } M_x \text{ does halt on x} \qquad \Leftrightarrow (M_x, x) \in K \end{cases}$$

In simpler terms, we want M' to terminate on programs with matching indices that do not halt:

$$M' \textbf{ halts on } x \textit{(i.e., returns 0)} \Leftrightarrow \textbf{ program } x \textbf{ does not halt on } x$$

However, this leads to a contradiction when we recognize that M' (in virtue of being a valid Turing Machine that we assume halts) has its own index, say e, and that it too **can be run on its own input** e. This gives rise to the following:

$$\begin{aligned} M_e \textbf{ halts on } e &\Leftrightarrow [\text{program } x] \textbf{ does not halt on } x && \textit{(definition)} \\ M_e \textbf{ halts on } e &\Leftrightarrow \qquad [M_e] \textbf{ does not halt on } e && \textit{(substitution with e)} \end{aligned}$$

the last of which is a clear contradiction. Translating this into an assertion about set membership, this amounts to saying

$$(M_e, e) \in K \Leftrightarrow (M_e, e) \notin K,$$

which is again a contradiction, thus making our assertion that K is decidable not tenable.[11] □

The main point here is that for any universal algorithm that tries to determine membership in K, there will inevitably be inputs for which deciding membership will give raise to a contradiction

[11] If Turing machines make you feel uncomfortable, without loss of generality we can switch again to thinking about functions where M_j in this case is a (computable) numeric function f_j over a single variable (e.g., $f_j(y) = y^2$). Following an example from Davis (1958), to conceptualize how an enumeration might be constructed (in principle, there are many ways of enumerating such functions), imagine that each f_j has an English description associated with it (e.g., $f_j(y) = y^2 \rightarrow_{\text{DESC}}$ *y squared*), and that we order such descriptions in terms of the number of letters in the their descriptions (in cases where descriptions have the same length, we can resolve this by ordering them alphabetically). Using this ordering, we can then assign numeric codes accordingly.

A simplified version of this result can be stated with these functions by considering a simple function $g(x)$ which, given an input x, adds 1 to the function indexed by x applied to x:

$$g(x) = f_x(x) + 1.$$

For example, imagine that $x = 3$ and that $f_3(y) = y^2$, then $g(3) = f_3(3) + 1 = 3^2 + 1 = 10$. Though g will be able to be computed for most inputs, the problem is that g has its own index e. If we run it on its own input we get the following problematic case: $g_e(e) = g_e(e) + 1$, which makes g not computable in the general case. On the basis of this, we can prove the undecidability of the Halting Problem.

that makes it impossible for any algorithm to decide whether or not it is in the set (and hence will cause an algorithm to run forever[12]). We note that the Halting Set as defined is not itself a set of numbers of the type we defined in Definitions 4-5 (i.e., a subset of $\mathbb{N}$), however it can be used to build an explicit set of numbers. The standard example is called the *diagonal halting set* defined as

$$K_0 = \Big\{ x \mid M_x(x) \downarrow \Big\}, \tag{8}$$

which can also be used as the basis of the proof above. We consider a different example set A that looks superficially closer the types of sets encountered in number theory, as described in the following corollary (which is the most important result in this section):

Corollary 1. *There exists recursively enumerable sets that are not recursive.*

Proof. We have already basically proven this with K, though will make the case again using a specific example set from Poonen (2008) to emphasize the larger point. Imagining again that we have an enumeration of Turing machines M_1, M_2, ..., consider the following set $A \subset \mathbb{N}$:

$$A = \Big\{ j = 2^x 3^y \mid M_x(y) \downarrow \Big\} \tag{9}$$

We will first establish A **is recursively enumerable**, which we can do by imagining the following (completely impractical) algorithm: using our enumeration of programs, we can loop through all numbers and pairs $x, y = 1, ..., \infty$ and execute each $M_x(y)$ in parallel[13]. In cases where $M_x(y) \downarrow$ is true, the computation has to stop after some finite number of steps, at which point we can simply print $2^x 3^y$ (this will potentially lead to repeats, though this is fine according to our definition). We can then simply ignore cases where the programs run forever, since these are clearly cases where $M_x(y) \downarrow$ is not true.

In terms of this set not being recursive, clearly deciding membership in A would require solving membership in K, which we have already shown is not possible given the undecidability of the Halting Problem (Theorem 2). Therefore, for some inputs the program might run forever, which we can safely ignore in this case. $\square$

[12] A more practically minded programmer might retort that this issue can be easily fixed by removing the cases that give rise to this contradiction. While this might work is some practical scenarios, it misses the point, which is that our notion of a Halting Set, which at first glance seems like a reasonable concept, is fundamentally problematics since it gives rise to such *bugs*.

[13] As before, this algorithm is wildly impractical, though still permitted under our theoretical notion of an algorithm. A brief anecdote related to this general theme: Yuri Matiyasevich, who ultimately solved Hilbert's 10th problem, was an occasional guest at my alma mater, the University of Stuttgart, and I distinctly remember the abstract for one of his talks in 2014:

> *Algorithm of Alfred Tarski for deciding the validity of a closed first-order formula with variables ranging over real numbers is one of the most difficult known decision procedures. A version of this algorithm will be presented with all details. This version is:*
>
> - *- easy to understand,*
> - *- easy to implement on a computer,*
> - *- **extremely inefficient** (emphasis mine).*

I don't recall the details of his particular algorithm, but generally when a mathematician says that an algorithm is inefficient, it is likely to be inefficient beyond all comprehension.

Given the relatively low bar, practically speaking, for demonstrating whether a set is recursively enumerable (i.e., we allow algorithms that look through infinite values of numbers and use infinite amounts of time and space), one might wonder whether there are any sets that are not recursively enumerable. To answer this, we have the following result (which might provide additional insight).

Lemma 3. *The complement of the set A from Corollary 1, $\overline{A}$, is not recursively enumerable.*

Proof. (very rough description) $\overline{A}$ being recursively enumerable (i.e., having an algorithm to enumerate its members) would imply that A is recursive (we could use such an algorithm for deciding $x \notin A$) and that the Halting Problem is decidable, in contradiction to what we have already proven. □

Number Theory Meets Computability

With the computer science background provided in the last section, we can now return to diophantine sets and rapidly state the main results. First, the following result is the first to relate diophantine sets and recursively enumerable sets and should be straightforward.

Theorem 3. *Any diophantine set of numbers is recursively enumerable.*

Proof. By definition, a set being diophantine means that we have a corresponding polynomial $p(a, x_1, ..., x_n)$. Using such a polynomial, an example (and again, highly impractical) algorithm for listing members of the set is one that simply enumerates all $n+1$ tuples of numbers in parallel and print a each time these numbers produce a 0 using the equation. □

Now for the most surprising result called the *Matiyasevich Theorem* (or the *Davis-Putnam-Robinson-Matiyasevich* theorem (DPRM), so-called after the overall group of researchers that laid much of the ground work for this result):

Theorem 4. *(*DPRM*) Any recursively enumerable set of numbers is diophantine.*

This immediately implies a negative solution to Hilbert's 10th problem (before looking at the proof below, spend a moment to reflect on the profoundness of this theorem!):

Corollary 2. *Hilbert's 10th problem is undecidable (i.e., no algorithm exists for solving arbitrary arbitrary diophantine equations).*

Proof. Given the DPRM theorem, every recursively enumerable set is diophantine (or has a corresponding equation $p(a, x_1, .., x_n) = 0$). It follows from this that the set A from Corollary 1 is diophantine. Hence, the existence of an algorithm to solve Hilbert's 10th problem would imply an algorithm for determining set membership in A (or any comparable set, such as the diagonal Halting set K_0 in Equation 8), which would then imply the decidability of the Halting Problem. □

Okay, so we now have now stated the *negative solution* to Hilbert's 10th problem. Of course, really understanding how we got here involves saying a few words about the DPRM theorem, which is the least trivial part of the entire result. Before we do this, however, let's just consider some of the remarkable consequences of the DPRM Theorem outside of the main result above.

Corollary 3. *The set of prime numbers is diophantine*[14]*, i.e., there exists a polynomial equation $p(a, x_1, ..., x_n) = 0$ in positive values that has a solution if and only if a is prime.*

While DPRM theorem wasn't proven until around 1970, it was conjectured to be true by Martin Davis dating back to the 1950's. Matiyasevich remarks that it was this corollary to the conjecture that led many mathematicians to doubt its validity (since prime numbers are known to have quite a random nature; the idea that they could be described in a single polynomial equation seemed far-fetched). Subsequent work based on DPRM led to the discovery of specific polynomial equations for primes, the most amusing of which is the following equation (which has 26 variables conveniently matching the number of letters in the alphabet):

Theorem 5. *The set of all prime numbers is equal to the set of all positive values k of the following polynomial (Jones et al., 1976):*

$$\begin{aligned}(k+2)\{1 &- [wx+h+j-q]^2\\
&- [(gk+2g+k+1)(h+j)+h-z]^2\\
&- [2n+p+q+z-e]^2\\
&- [16(k+1)^3(k+2)(n+1)^2+1-f^2]^2\\
&- [e^3(e+2)(a+1)^2+1-o^2]^2\\
&- [(a^2-1)y^2+1-x^2]^2\\
&- [16r^2y^4(a^2-1)+1-u^2]^2\\
&- [n+l+v-y]^2\\
&- [((a+u^2(u2-a))^2-1)(n+4dy)^2+1-(x+cu)^2]^2\\
&- [(a^1-1)l^2+1-m^2]^2\\
&- [q+y(a-p-1)+s(2ap+2a-p^2-2p-2)-x]^2\\
&- [z+pl(a-p)+t(2ap-p^2-1)-pm]^2\\
&- [ai+k+1-l-i]^2\\
&- [p+l(a-n-1)+b(2an+2a-n^2-2n-n)-m]^2\}\end{aligned}$$

assuming non-negative values for all variables.

This equation requires some explanation, since it clearly deviates from the standard diophantine form we have been considering. It involves an alternative way of describing polynomial equations that was first observed by Hilary Putnam in Putnam (1960). Given any diophantine set $S \subseteq \mathbb{N}$ (represented by a polynomial $p(a, x_1, ..., x_n)$ with natural number variables), such a set has a corresponding polynomial in natural numbers $q(x_0, ..., x_n)$ where

$$q(x_0, ..., x_n) = x_0\big(1 - p^2(x_0, ..., x_n)\big), \tag{10}$$

[14]This is in virtue of the prime numbers being recursively enumerable.

and hence the range of that polynomial includes exactly the numbers a that solve p (i.e., the factor $1 - p^2$ must be positive, which is only possible when p evaluates to 0). Therefore, it is possible to offer an equation in the form shown above and to express Corollary 3 in the following way:

Corollary 4. *There exists some polynomial equation $q(x_1, ..., x_n) = a$ which has a solution if and only if a is a prime number.*

As remarked in Poonen (2008), to prove DPRM theorem the authors *essentially built a computer out of diophantine equations; the proof ... looks curiously like the construction of a complicated computer program, with high-level routines built out of more elementary ones, except that instead of routines one has diophantine equations everywhere.* One consequence of DPRM is that it becomes possible to construct *universal diophantine equations* that can simulate Universal Turing Machines (or other equivalent (universal) models of computation)!

Regarding the questions asked in the beginning about whether particular equations have a fixed number of solutions (or no solutions), coming up with a universal algorithm here also turns out to be undecidable as a consequence of DPRM (for details see Davis (1972)).

Getting to the DPRM Theorem (rough outline and history) The DRPM Theorem is a truly remarkable result that had nonetheless been anticipated for several decades before its final resolution (recall again Emil Post's ominous conjecture 25 years before that Hilbert's 10th problem *begs for an unsolvability proof*). A closely related result was obtained by Davis, Putnam and Robinson (Davis et al., 1961) for *exponential diophantine equations* (i.e., diophantine equations extended to include *exponentials* in combination with addition and multiplication, as permitted in *ordinary* diophantine equations[15]).

Theorem 6. *Every recursively enumerable set has an* exponential Diophantine representation *of the form:*

$$S = \Big\{a \mid \exists x_1, ..., x_n[p_{E_1}(a, x_1, ..., x_m) = p_{E_2}(a, x_1, ..., x_n)]\Big\}$$

for exponential diophantine equations p_{E_1} and p_{E_2}.

This immediately led to a negative solution to Hilbert's 10th problem for exponential diophantine equations. As Matiyasevich often cites in his lectures, this result was met with the following criticism when related to the larger Hilbert problem, in this case by one reviewer of their article in 1962 for the *Mathematical Review*:

> ...**These results are superficially related to Hilbert's tenth Problem** on (**ordinary**, i.e., non-exponential) Diophantine equations. The proof the authors' result, though very elegant, does not use recondite facts in the theory of numbers nor in the theory of r.e. [recursively enumerable] sets, and so it is likely that the **present result is not closely connected with Hilbert's tenth Problem**. Also, it is not altogether plausible that all (ordinary) Diophantine problems are uniformly reducible to those in a fixed number of variables of fixed degree, which would be the case if all r.e. sets were diophantine.

[15]This includes Fermat's equation considered in Equation 2, as well as more unusual equations such as $2x^{3y^x z + x^2} = 5x^2 + yz$ (Poonen, 2008).

In other words, exponential diophantine equations were not exactly what Hilbert had in mind when he formulated the problem, hence this result did not suffice to provide a negative solution to the original problem (At the end of this comment, you can also see the reviewers' skepticism about all recursively enumerable sets being diophantine). Nonetheless, Matiyasevich writes that the significance and relevance of this result to the larger problem was overlooked by many people beyond this single reviewer, and even included his own advisor at the time who had initiated his interest in Hilbert's 10th problem[16].

Building on the result above, the missing link involved proving that exponentiation is diophantine, or that the following set is diophantine[17]:

$$\left\{(a,b,c) \in \mathbb{N}^3 \mid c = a^b\right\}$$

by showing that it has a corresponding 3 variable diophantine equation $p_a(a,b,c,x_1,...,x_n) = 0$ (which one could use to transform any exponential diophantine equation into an ordinary diophantine equation, albeit with some additional variables). Julia Robinson had earlier proved some sufficient conditions for p_a to exist, namely that it would suffice to find a particular 2 variable diophantine equation $p_b(a,b,x_1,...,x_n)$ that exhibits *exponential growth*. Building on recent work by Nikolai Vorobyov, Matiyasevich uses properties of Fibonacci numbers (F_n) to prove that the following set is diophantine:

$$\left\{(a,b) \mid a > 0, b = F_{2a}\right\}$$

which ultimately leads to DPRM (for full details of the proof, we again urge readers to consult Davis (1973) and Matiyasevich (1993)).

In terms of the larger theme, here we can see more clearly the curious way in which number theory is being studied under this approach: rather than starting an explicit diophantine equation and trying to prove properties of that equation (as ordinarily done in number theory), we are instead starting with a set then try to find an diophantine equation that characterizes that set. If such an equation is found, we can then say something about its properties by deferring to what we know about sets from computability theory.

References

A. R. Booker. Cracking the problem with 33. *Research in Number Theory*, 5(3):26, 2019.

A. Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.

M. Davis. *Computability and Unsolvability*. Dover Publications (Dover Edition), 1958.

[16]He recounts in Matijasevich (1992) the following dialogue: *I asked my scientific adviser, Sergel [sic] Maslov, what to do next. He answered: 'Try to prove the algorithmic unsolvability of Diophantine equations. This problem is known as Hilbert's tenth problem, but that does not matter to you.' – 'But I haven't learned any proof of the unsolvability of any decision problem.' – 'That also does not matter. Unsolvability is nowadays usually proved by reducing a problem already known to be unsolvable to the problem whose unsolvability one needs to establish, and you understand the technique of reduction well enough.' – 'What should I read in advance?' – 'Well, there are some papers by American mathematicians about Hilbert's tenth problem, but you need not study them.' – 'Why not?' – 'So far the Americans have not succeeded, so their approach is most likely inadequate.'*

[17]Assuming a slightly more complex definition of a diophantine set than so far considered, which in particular permits sets of $m-$tuples of $\mathbb{N}^m$ rather than only subsets of $\mathbb{N}$; see again Footnote 7.

M. Davis. On the number of solutions of diophantine equations. *Proceedings of the American Mathematical Society*, 35(2): 552–554, 1972.

M. Davis. Hilbert's Tenth Problem is Unsolvable. *The American Mathematical Monthly*, 80(3):233–269, 1973.

M. Davis, H. Putnam, and J. Robinson. The decision problem for exponential diophantine equations. *Annals of Mathematics*, pages 425–436, 1961.

J. P. Jones and Y. V. Matijasevič. Proof of recursive unsolvability of hilbert's tenth problem. *The American Mathematical Monthly*, 98(8):689–709, 1991.

J. P. Jones, D. Sato, H. Wada, and D. Wiens. Diophantine Representation of the Set of Prime Numbers. *The American Mathematical Monthly*, 83(6):449–464, 1976.

Y. Matijasevich. My collaboration with julia robinson. *The Mathematical Intelligencer*, 14(4):38–45, 1992.

Y. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.

H. Pasten. Diophantine equations and why they are hard. 2019. URL https://imaginary.org/sites/default/files/snapshots/snapshots-2019-003.pdf.

B. Poonen. Undecidability in Number Theory. *Notices of the AMS*, 55(3), 2008.

H. Putnam. An unsolvable problem in number theory. *The Journal of Symbolic Logic*, 25(3):220–232, 1960.

P. Smith. The mrdp theorem. 2011. URL https://www.logicmatters.net/resources/pdfs/MRDP.pdf.

A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.