These lecture notes cover Hilbert's Tenth Problem. They are intended for the students taking the module *MA3J9-Historical Challenges in Mathematics* at the University of Warwick. We follow very closely the notes of a talk given by Yuri Matiyasevich that can be found here.

If you have any comments or find any mistakes, please let me know, either in person or by email (`d.testa` at `warwick.ac.uk`).

Damiano Testa
January 2020

Hilbert's Tenth Problem: Solvability of Diophantine equations

Find an algorithm that, given a polynomial $D(x_1, \ldots, x_n)$ with integer coefficients and any number of unknowns decides whether or not there are integers $a_1, \ldots, a_n \in \mathbb{Z}$ such that $D(a_1, \ldots, a_n) = 0$.

1 Introduction

At the time when Hilbert posed the question, computers and algorithms did not exist. Thus, the statement above is a more recent reformulation of the actual question. Already in Hilbert's times, though, there was an extensive list of results proving that special polynomial equations with integer coefficients admit or do not admit integer solutions.

$$\underset{\text{Pell's equation}}{x^2 - 2y^2 = -1}, \qquad \underset{\text{Fermat's equation}}{x^5 + y^5 = z^5}, \qquad \underset{\text{sum of three cubes}}{x^3 + y^3 + z^3 = 114}. \tag{1}$$

Examples of Diophantine equations

The pair $(x, y) = (1, 1)$ is an integer solution to the first equation in (1). All integer solutions of the second equation in (1) must have $xyz = 0$. As of January 2020, it is unknown whether the third equation in (1) admits an integer solution or not.

Definition 1.1. A *Diophantine equation* is an equation of the form

$$D(x_1, \ldots, x_n) = 0,$$

where D is a polynomial in some number n of variables and integer coefficients.

The approaches to finding solutions to Diophantine equations usually rely on widely different methods, each adapted to the particular shape of the equation being solved. Hilbert thought of specializing the question to the following decision problem: Given a Diophantine equation, decide if it is soluble or not in integers. He was interested in finding an algorithm that would answer the decision problem *uniformly* for all Diophantine equations.

Through the efforts of several mathematicians (Davis, Putnam, Robinson, Matiyasevich, among others) over the years, it was discovered that the algorithm sought by Hilbert cannot exist.

Theorem 1.2 (Undecidability of Hilbert's Tenth Problem)**.** *There is no algorithm which, for a given arbitrary Diophantine equation, would tell whether the equation has a solution or not.*

Before giving some steps in the proof of this result, we give some context. There are several variants of this question. We may want the set of coordinates of the required solutions to be

- the integers $\mathbb{Z}$ (Hilbert's original question);
- the natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$;
- the rational numbers $\mathbb{Q}$;
- the real numbers $\mathbb{R}$;
- the complex numbers $\mathbb{C}$;

and so on. Similarly, we may be interested in polynomial equations with a fixed number of variables, of a fixed degree, with coefficients in a number field, in a finite field, in the field of real numbers, ... Of course, the answers to these questions are not entirely unrelated. Some of these decision questions are easy to answer, others are notorious open problems.

2 From the integers to the natural numbers

We emphasize that, for this module, the set of *natural numbers* $\mathbb{N}$ consists of the *non-negative* integers and *includes* 0.

In what follows, we talk about a *computer*, when we really mean a *Turing machine*. A *Turing machine* is a device that, given a finite list of instructions and arbitrary access to time and memory, performs calculations and outputs values. A *universal Turing machine* is a Turing machine that can reproduce the output of any finite list of rules read by any other Turing machine. Essentially all real-world computers (or even phones or watches!) are universal Turing machines, provided we imagine that they have access to an arbitrarily large supply of memory and time.

We use the expression *computer program* or *procedure* for a finite sequence of instructions that a computer uses to run, possibly forever, while computing values. The set, possibly empty, of computed values, may involve repetitions. We reserve the term *algorithm* for a procedure that terminates on all possible inputs.

We introduce a little bit of notation, to simplify our treatment. Let $R \subset \mathbb{C}$ be a subset. Typically, we choose $R \in \{\mathbb{Z}, \mathbb{N}\}$. We denote by $\exists D(R)$ (standing for "there exists a decision algorithm for Diophantine equations and solutions in R") the statement

$$\exists D(R)\colon \left(\begin{array}{ll} & \text{There is an algorithm with} \\ \texttt{input} & \text{a polynomial } D(x_1, \ldots, x_n) \text{ with integer coefficients} \\ & \text{in any number of variables.} \\ \texttt{output} & \begin{cases} \texttt{true}, & \text{if there are } a_1, \ldots, a_n \in R \text{ such that} \\ & D(a_1, \ldots, a_n) = 0; \\ \texttt{false}, & \text{otherwise.} \end{cases} \end{array} \right)$$

To uniformise with the standard notation used in the resolution of Hilbert's question, we restrict our attention to *non-negative* integers solving Diophantine equations. This is not a serious restriction, and covers Hilbert's question, as we will see shortly. First, recall Lagrange's Four-Square Theorem.

Theorem 2.1 (Lagrange's Four-Square Theorem)**.** *Every non-negative integer is the sum of four squares of integers.*

We can now prove our first reduction.

Lemma 2.2. *The two statements $\exists D(\mathbb{Z})$ and $\exists D(\mathbb{N})$ are equivalent.*

This means that the decision problem with *integer* solutions is equivalent to the decision problem with *non-negative integer* solutions.

Proof. Suppose that $\exists D(\mathbb{Z})$ holds and let $A_{\mathbb{Z}}$ be the implied decision algorithm for integer solutions. Thus, for a polynomial $D(x_1, \ldots, x_n)$ with integer coefficients, $A_{\mathbb{Z}}(D)$ returns `true`, if the equation $D = 0$ has a solution with integer coordinates, and `false`, otherwise.

We want to show that there is an algorithm $A_{\mathbb{N}}$ such that for every polynomial $D(x_1, \ldots, x_n)$ with integer coefficients, $A_{\mathbb{N}}(D)$ returns `true`, if the equation $D = 0$ has a solution with *non-negative* integer coordinates, and `false`, otherwise. For this, we use Lagrange's Four-Square Theorem. For each $i \in \{1, \ldots, n\}$, we introduce the 4 new variables $y_{i,0}, y_{i,1}, y_{i,2}, y_{i,3}$ and we replace the ith variable x_i by the substitution

$$x_i \quad \longmapsto \quad y_{i,0}^2 + y_{i,1}^2 + y_{i,2}^2 + y_{i,3}^2.$$

In this way, the polynomial $D(x_1, \ldots, x_n)$ with integer coefficients in n variables becomes the polynomial

$$D_y = D(y_{1,0}^2 + y_{1,1}^2 + y_{1,2}^2 + y_{1,3}^2, \ldots, y_{n,0}^2 + y_{n,1}^2 + y_{n,2}^2 + y_{n,3}^2)$$

still with integer coordinates, but in $4n$ variables. We now run the algorithm $A_{\mathbb{Z}}$ on D_y. As the y-variables range among the integers, the corresponding sums of four squares range over exactly the set of non-negative integers, by Lagrange's Four-Square Theorem. It follows that $A_{\mathbb{Z}}(D_y)$ returns `true` if and only if the polynomial D has a non-negative integer solution, as required.

To prove the converse, assume that $\exists D(\mathbb{N})$ holds and let $B_{\mathbb{N}}$ be the implied decision algorithm for non-negative integer solutions. Thus, for a polynomial $D(x_1, \ldots, x_n)$ with integer coefficients, $B_{\mathbb{N}}(D)$ returns `true`, if the equation $D = 0$ has a solution with non-negative integer coordinates, and `false`, otherwise.

Let $D(x_1, \ldots, x_n)$ be a polynomial with integer coefficients. Run the algorithm $B_{\mathbb{N}}$ on the product

$$D_{\pm} = \prod_{\epsilon_1, \ldots, \epsilon_n \in \{\pm 1\}} D(\epsilon_1 x_1, \ldots, \epsilon_n x_n).$$

We obtain that $B_{\mathbb{N}}(D_{\pm})$ returns `true` if and only if there are signs $\sigma_1, \ldots, \sigma_n \in \{\pm 1\}$ and natural numbers $a_1, \ldots, a_n \in \mathbb{N}$ such that $D(\sigma_1 a_1, \ldots, \sigma_n a_n)$ vanishes.

This precisely means that $B_{\mathbb{N}}(D_{\pm})$ returns `true` if and only if there are integers $b_1, \ldots, b_n \in \mathbb{Z}$ such that $D(b_1, \ldots, b_n)$ vanishes, as required. □

From now on, unless otherwise stated, we focus on the decision problem $\exists D(\mathbb{N})$ for non-negative integer solutions to Diophantine equations. As we saw, the statements $\exists D(\mathbb{Z})$ (Hilbert's Tenth Problem) and $\exists D(\mathbb{N})$ are equivalent, yet our treatment is going to be slightly simplified talking about $\exists D(\mathbb{N})$.

> **Aside.** While the statements $\exists D(\mathbb{N})$ and $\exists D(\mathbb{Z})$ are equivalent (and algorithmically undecidable), it is unknown whether the statement $\exists D(\mathbb{Q})$ is algorithmically decidable or not.

As an initial approach to trying to find a decision algorithm for Diophantine equations, one could try the following. Let $D(x_1, \ldots, x_n)$ be a polynomial with integer coefficients in n variables.

- Enumerate all n-tuples $(a_1, \ldots, a_n) \in \mathbb{N}^n$ in any way.

  For instance, for each natural number t, we could take the finitely many n-tuples $(a_1, \ldots, a_n)$ with sum $a_1 + \cdots + a_n = t$, and order them lexicographically.

- For each n-tuple $\underline{a}$ in the list, successively compute the evaluation $D(\underline{a})$.

- If the evaluation $D(\underline{a})$ equals 0, then return `true`: we have proved that there is a solution in natural numbers to the Diophantine equation $D = 0$.

- If the evaluation $D(\underline{a})$ does not equal 0, then move on to the next n-tuple and back to the evaluation step.

The procedure just described has the following properties. If, on the one hand, the input Diophantine equation $D = 0$ admits a solution with non-negative integer coordinates, then our procedure will eventually stumble upon such a solution. Once that happens, the procedure terminates, giving `true` as output: we know that the input D is soluble. If, on the other hand, the input equation $D = 0$ does not admit a solution with non-negative integer coordinates, then our procedure will never terminate: it will eventually try each n-tuple and will check that the value of D is never 0, but it will not terminate in a finite time. For an easy example, if the input were the constant polynomial $D(x_1, \ldots, x_n) = 1$, then the procedure will never terminate, as the evaluations are always equal to 1.

This very elementary procedure almost resolves Hilbert's Tenth Problem. All that is missing, is, in the case of a Diophantine equation with no solution, a criterion for deciding when we have tried enough n-tuples. While it may feel like we have almost solved our problem, it turns out that filling in this apparently tiny, gap is precisely where all the difficulty is hiding.

3 Diophantine and listable sets: definition

Let us take a small digression to see a strategy to resolve Hilbert's Tenth Problem and show that it is undecidable. For any individual Diophantine equation $D = 0$, the existence of an algorithm deciding if it is soluble or not is completely clear: among the two algorithms, one that always returns `true` and the other that always returns `false`, one (and only one) decides correctly the solubility of $D = 0$. Of course, we would not know which of the two algorithms to choose. Nevertheless, the *existence* of an algorithm correctly deciding any given Diophantine equation, or any finite number of them, is clear! It is also clear that this kind of "solution" is unsatisfactory and we would like a better answer. To avoid these issues, we are going to look for an infinite number of Diophantine equations $D_t = 0$, one for each natural number $t \in \mathbb{N}$, for which there is no decision algorithm. So far, this is not really a strengthening of our initial problem: the set of all Diophantine equations (in a fixed, countable set of variables) is countable and the parameter t introduced above could simply be an enumeration of all Diophantine equations. Thus, we view a Diophantine equation $D(t, x_1, \ldots, x_n) = 0$ in $n+1$ variables as a family of Diophantine equations, one for each value of t. We limit our choice of families to families of this form: it will turn out that this is enough for our purposes. Such families allow us to define special subsets of $\mathbb{N}$, obtained as the values $a \in \mathbb{N}$ for which the corresponding Diophantine equation $D(a, x_1, \ldots, x_n) = 0$ admits a solution. For added flexibility, we allow families depending on more than just one parameter.

Definition 3.1. A subset $S \subset \mathbb{N}^r$ is *Diophantine* if there is a non-negative integer n and a polynomial $D(t_1, \ldots, t_r, x_1, \ldots, x_n)$ with integer coefficients such that

$$S = \left\{ (t_1, \ldots, t_r) \in \mathbb{N}^r \;\middle|\; \begin{array}{l} \exists (x_1, \ldots, x_n) \in \mathbb{N}^n, \\ D(t_1, \ldots, t_r, x_1, \ldots, x_n) = 0 \end{array} \right\}.$$

In words, a Diophantine set is the set of values of the parameters of some Diophantine equation for which the corresponding Diophantine equation admits solutions. As always, we require all coordinates to be non-negative integers.

We now introduce a closely related (indeed, equivalent) concept that will guide us in our approach to understanding Hilbert's Tenth Problem and undecidability.

Definition 3.2. A subset $L \subset \mathbb{N}^r$ is *listable* if there is a computer program that, when left to run indefinitely,

- only outputs r-tuples in L;
- eventually outputs all the elements of L, possibly with repetitions.

Most of the explicit examples of listable sets that we will see are in fact subsets of $\mathbb{N}$. Nevertheless, having the freedom of allowing r-tuples, will simplify our treatment. It is easy to come up with examples of listable sets in $\mathbb{N}$.

Example 3.3. The following subsets of $\mathbb{N}$ are all listable:

- the set of all natural numbers $\mathbb{N}$,
- the set of odd integers greater than or equal to $2^{132007} + 7^{315}$,
- the set of squares $\{n^2 : n \in \mathbb{N}\}$,
- the set of prime numbers different from $2^{82,589,933} - 1$.

We point out that in the examples just given, not only each set is listable, but also its complement in $\mathbb{N}$ is listable. This is simply a consequence of the fact that we did not try too hard to write a complicated example! Indeed, the existence of listable subsets of $\mathbb{N}$ with non-listable complement is one of the two main ingredients in the proof of undecidability of Hilbert's Tenth Problem.

From our point of view, we can therefore isolate three kinds of subsets of $\mathbb{N}$:

(LL) *listable* sets $S \subset \mathbb{N}$ with *listable* complement $\mathbb{N} \setminus S$;

(LN) *listable* sets $S \subset \mathbb{N}$ with *non-listable* complement $\mathbb{N} \setminus S$;

(NN) *non-listable* sets $S \subset \mathbb{N}$ with *non-listable* complement $\mathbb{N} \setminus S$.

Aside. We have seen examples of sets of type (LL), listable with listable complement. We will see later an example of a set of type (LN), listable with non-listable complement. For completeness, we show that sets of type (NN), non-listable with non-listable complement, do exist. This is a straightforward cardinality argument. A computer program is a finite list of finitely many symbols (satisfying certain grammatical and syntactical rules). Let A be the set of symbols that we are allowed to use. For example, A could consist of the 256 characters of the extended ASCII table, or the 137,929 Unicode characters. A computer program is therefore a finite list of symbols extracted from the set A. Thus, the set of all computer programs is contained in the union $\cup_{n\in\mathbb{N}} A^n$. As this is a countable union of finite sets, we deduce that there exist at most countably many computer programs. Since a computer program produces at most one listable set, we deduce that there are countably many listable sets. Thus, there are also countably many sets whose complement is listable. Overall, we deduce that there are countably many sets $S \subset \mathbb{N}$ with the property that at least one among S and $\mathbb{N} \setminus S$ is listable. Since there are uncountably many subsets of $\mathbb{N}$, we deduce that there exist (plenty of) sets that are neither listable, nor their complement is listable. An imprecise, but possibly intuitive, definition of a non-listable set with non-listable complement is as follows. Start with an empty set S. For each natural number n, flip a coin: if heads comes out, add n to the set S; if tails comes out, do not add it. The set obtained as an outcome of this infinite procedure is, almost surely, a non-listable set with non-listable complement.

Let us go back to our discussion. To approach Hilbert's Tenth Problem

- we show that there are listable sets with non-listable complement;
- we argue that Diophantine sets and listable sets coincide.

Assuming these two facts, the undecidability of Hilbert's Tenth Problem follows. Indeed, let $U \subset \mathbb{N}^r$ be a listable set with non-listable complement. Choose any Diophantine equation $D_U = 0$, depending on r parameters (and of course some number of variables), for which the set of values of the parameters giving rise to a soluble Diophantine equation is precisely U. The existence of D_U is a consequence of the fact that Diophantine and listable sets coincide. If there were an algorithm deciding, for each value of the parameters in $\mathbb{N}^r$, whether or not the Diophantine equation $D_U = 0$ were soluble, then there would be a computer program listing all the non-soluble values of $D_U = 0$. This means that $\mathbb{N}^r \setminus U$ would be listable, contrary to the assumption on U.

> **Aside.** To argue undecidability, it would be enough to show that there exists a Diophantine set whose complement is not listable. In principle, showing the equality of Diophantine and listable sets is therefore not necessary. Nevertheless, we will outline a proof of the equality of Diophantine and listable sets.

4 Listable sets with non-listable complement

We present here a construction of a listable subset $T \subset \mathbb{N}$ with the property that its complement $\mathbb{N} \setminus T$ is not listable.

Let us take a computer and let us create a list of programs that can run on the computer. We are interested in listing all programs that take natural numbers as input. This can be achieved in multiple ways. For instance, a computer program is, at its core, a (possibly very long) string of binary digits and we may associate to each computer program the corresponding binary number. Thus, we can take our computer, feed it successively each binary number i and see if the result is a computer program with $\mathbb{N}$ as an input set. (Usually, we would use a *compiler* to convert the binary number into machine-readable code.) If the resulting program has input set $\mathbb{N}$, then we add the binary number i to our list of programs, otherwise, we do not. The specifics of how we choose to produce such a list is irrelevant for our discussion: the only important property is that such lists *exist* and we can compute one. Thus, the set of binary numbers encoding computer programs is listable. Notice that most computer programs in our list will be very boring: some will immediately terminate, regardless of the input, others will waste longer and longer time and memory computing irrelevant values to eventually print a fixed number, such as `10`, others still will enter into a loop and never output anything. Nevertheless, for each natural number $i \geq 0$, we can talk about the ith program in our list.

We now sort all of our programs in two disjoint sets T and O: define

$$\begin{aligned} T &= \big\{i\colon i\text{th program terminates on input } i\big\} &&\subset \mathbb{N};\\ O &= \big\{i\colon i\text{th program does not terminate on input } i\big\} &&\subset \mathbb{N}. \end{aligned}$$

Notice the self-referential nature of this definition: we ask whether the computer program with index i terminates on input its own index i.

Clearly, the sets T and O partition the natural numbers: we have $T \cup O = \mathbb{N}$ and $T \cap O = \emptyset$.

Lemma 4.1. *The set T is listable and its complement O is not listable.*

Proof. **Step 1: the set T is listable.** To see this, we define a procedure, whose eventual outputs are precisely the indices i of computer programs in our list that terminate on input i.

(1) Start by defining a set V, initially empty, and a counter j, initially set to equal 0.

(2) Loop through all the computer programs with indices $i \leq j$, allowing the ith program to run with input its own index i for up to j steps. If one of these programs terminates, add the corresponding index to the set V.

(3) Once the loop terminates, increase the counter j by 1 and return to the loop in (2), with the higher value of j.

Provided we wait long enough, every computer program in our list will have a chance to run for an arbitrary amount of time on the relevant input. If the ith computer program ever terminates with input i, it will do so in finitely many steps, say in s steps. Therefore, the procedure described above adds the index i to the set V before the counter j exceeds $\max\{i, s\}$. Conversely, if the ith computer program does not terminate on input i, then the procedure described above will never output i. We deduce that the partial sets V produced by the program are always contained in T and eventually exhaust T. We obtain that T is listable: we even outlined a computer program listing T.

Step 2: the set $O = \mathbb{N} \setminus T$ is *not* listable. Proceed by contradiction and suppose that the set O is listable. Thus, we have two listing procedures:

- one, following from Step 1, with output the indices $i \in T$, i.e. the indices for which the corresponding computer program terminates with input i,

and

- the other with output the indices $i \in O = \mathbb{N} \setminus T$, i.e. the indices for which the corresponding computer program does not terminate with input i.

Define a program c as follows: on input $i \in \mathbb{N}$, alternatively use the listing programs for the set T and the set O, until one of the two outputs i. If we find that i belongs to T, then c enters into an infinite loop, never terminating. If we find that i belongs to O, then c terminates returning `0`. Informally, the program c terminates on input i if the ith program does not terminate with input i, and, viceversa, it does not terminate on input i if the ith program terminates with input i. As c is a computer program taking a natural number as input, it appears in our list of computer programs and hence has an index i_c.

We now ask ourselves whether the index i_c of the program c belongs to T or O. If it belonged to T, then, by definition of T, the i_cth program, namely c itself, terminates on input i_c. By definition of the program c, this means that the program c does not terminate on input i_c, and we reach a contradiction. The remaining possibility is that i_c belongs to O, so that the i_cth program,

again c, does not terminate on input i_c. Yet, the definition of c then forces c to terminate on input i_c and we reach again a contradiction. It follows that the computer program c cannot exist and, hence, that there cannot exist a listing procedure for the elements of O. □

5 Diophantine and listable sets: equivalence

So far, we showed that listable sets need not have listable complement. In order to prove undecidability of Hilbert's Tenth Problem, it suffices to find a Diophantine set whose complement is not listable. After the combined efforts of many mathematicians, over many years, Davis conjectured that the asymmetry between Diophantine and listable sets is only apparent.

Conjecture 5.1 (Davis)**.** *Every listable set is Diophantine.*

Informally, we can interpret this conjecture as saying that every computer program that can ever be written, whose output are r-tuples of natural numbers can be "mimicked" by a Diophantine equation. The mimicking is of course in the sense that, for each computer program which outputs r-tuples of natural numbers, there is a polynomial D in r parameters, plus an auxiliary number n of variables, such that the set of values computed by the program is precisely the set of values of the parameters for which the Diophantine equation $D = 0$ admits a solution.

One of the two implication is straightforward (see Lemma 5.2). For the converse, we will go a little bit into the details of how Diophantine sets can be interpreted as computer programs.

Lemma 5.2. *Every Diophantine set is listable.*

Proof. The argument is a simple modification of our initial (failed) approach at deciding Diophantine equations. Let $S \subset \mathbb{N}^r$ be a Diophantine set. Suppose that $D(t_1, \ldots, t_r, x_1, \ldots, x_n)$ is a polynomial with integer coefficients such that the identity

$$S = \left\{ (t_1, \ldots, t_r) \in \mathbb{N}^r \;\middle|\; \begin{array}{l} \exists (x_1, \ldots, x_n) \in \mathbb{N}^n, \text{ with} \\ D(t_1, \ldots, t_r, x_1, \ldots, x_n) = 0 \end{array} \right\}$$

holds.

Enumerate all $(r+n)$-tuples of natural numbers and evaluate D in succession on each one of them. If ever we run into an $(r+n)$-tuple $(t_1, \ldots, t_r, x_1, \ldots, x_n)$ in $\mathbb{N}^{r+n}$ solving the equation $D(t_1, \ldots, t_r, x_1, \ldots, x_n) = 0$, then we output the r-tuple $(t_1, \ldots, t_r)$. In either case, we continue.

Clearly, we only output elements in S during our procedure (possibly multiple times). Moreover, every r-tuple in S will eventually appear as an output, since the witness solution $(t_1, \ldots, t_r, x_1, \ldots, x_n)$ of $D = 0$ showing that $(t_1, \ldots, t_r)$ is an element of S, appears at some point of our enumeration of all $(r + n)$-tuples of natural numbers. This concludes the proof that every Diophantine set is listable. □

We now turn to the converse, namely showing that every listable set is Diophantine. We only partially show this implication. The strategy is to establish a connection between Diophantine sets and computer programs and to argue that all computer programs arise via this connection.

Indeed, we view a Diophantine equation $D(t_1, \ldots, t_r, x_1, \ldots, x_n) = 0$ in r variables and n unknowns as a "coding" of a computer program outputting r-tuples of natural numbers. The program coded by D is the program described in the proof of Lemma 5.2. We enumerate all $(r+n)$-tuples of natural numbers, successively evaluating D at each $(r+n)$-tuple and, when D evaluates to 0 on the $(r+n)$-tuple $(a_1, \ldots, a_r, y_1, \ldots, y_n)$, the program outputs the r-tuple $(a_1, \ldots, a_r)$.

The way to reduce a computer operation to a Diophantine set, proceeds via the "graph" of the corresponding operation. For instance, suppose that we want to find a Diophantine set encoding the statement that the natural numbers a, b satisfy the inequality $a \leq b$. Thus, we try to prove that the set $M \subset \mathbb{N}^2$ consisting of all pairs $(a, b) \in \mathbb{N}^2$ satisfying $a \leq b$ is Diophantine. We achieve this using the equivalence

$$a \leq b \quad \Longleftrightarrow \quad (\exists x \in \mathbb{N})\colon b = a + x.$$

Hence, the relation $\leq$ on natural numbers is encoded by the Diophantine equation $b - a - x = 0$, with two parameters (a, b) and one variable x (recall that a, b, x are non-negative!). Similarly, for divisibility, the equivalence

$$a \mid b \quad \Longleftrightarrow \quad (\exists x \in \mathbb{N})\colon b = ax$$

shows that the divisibility relation on natural numbers is encoded by the Diophantine equation $b - ax = 0$, with two parameters (a, b) and one variable x. To see an example involving more parameters, we analyse the congruence relation $a \equiv b \pmod{c}$. In this case, a possible equivalence is

$$a \equiv b \pmod{c} \quad \Longleftrightarrow \quad (\exists x \in \mathbb{N})\colon \big((a = b + cx) \text{ or } (a = b - cx)\big).$$

In this case, besides having one more parameter c, we also have to come up with a way of encoding the logical operator `or`. To deal with this, we use that the product of two natural numbers vanishes if and only if at least one of the factors vanishes: this clearly has the same logical value as the operator `or`. Therefore, the set of triples (a, b, c) such that the relation $a \equiv b \pmod{c}$ holds is the same as the Diophantine set defined by the equation

$$(a - b - cx)(a - b + cx) = 0,$$

in the three parameters a, b, c and the variable x. Let us see an example with the `and` operator. Suppose that we want to encode as a Diophantine set, the relation on triples (a, b, c) satisfying $a \leq \min\{b, c\}$, that is the conjunction of the two relations $a \leq b$ `and` $a \leq c$. As before, we have the equivalence

$$a \leq \min\{b, c\} \quad \Longleftrightarrow \quad \Big(\big((\exists x \in \mathbb{N})\colon b = a + x\big) \texttt{ and } \big((\exists x \in \mathbb{N})\colon c = a + x\big)\Big).$$

Now, we exploit that two natural (or even real) numbers are both zero if and only if the sum of their squares vanishes. This allows us to encode the relation $a \leq \min\{b, c\}$ as the Diophantine equation

$$(b - a - x)^2 + (c - a - y)^2 = 0$$

in three parameters a, b, c and *two* variables x, y. We used two variables, since, if we used only one variable x and wrote the Diophantine equation

$$(b - a - x)^2 + (c - a - x)^2 = 0,$$

the triples (a, b, c) for which there is a solution are precisely the triples with $a \leq b$ and $b = c$.

The equivalence between Diophantine and listable sets can be obtained by giving an explicitly "dictionary" allowing us to convert every computer program P having r-tuples of natural numbers as outputs into a Diophantine equation $D_P(t_1, \ldots, t_r, x_1, \ldots, x_n) = 0$ in r-parameters and some additional number n of variables, such that the equality

$$\left\{ \begin{array}{c} \text{outputs of the} \\ \text{program } P \end{array} \right\} = \left\{ \begin{array}{c} \text{parameters } (a_1, \ldots, a_r) \in \mathbb{N}^r \text{ for which} \\ \text{the Diophantine equation} \\ D_P(a_1, \ldots, a_r, x_1, \ldots, x_n) = 0 \text{ admits a} \\ \text{solution in natural numbers.} \end{array} \right\}$$

holds.

We already saw a few examples of Diophantine (and hence listable) sets. To get a better feeling for what kind of issues arise, let us see two further examples.

Example 5.3 (Composite numbers)**.** Let $C = \{4, 6, 8, 9, 10, \ldots\} \subset \mathbb{N}$ be the set of composite natural numbers, that is, the numbers that are a product of two (not necessarily distinct) natural numbers, each at least equal to 2. This suggests the following equality

$$C = \Big\{ t \in \mathbb{N} \colon\ (\exists x, y \in \mathbb{N}),\ t = (x + 2)(y + 2) \Big\}.$$

Thus, the set C is Diophantine, and a possible Diophantine equation describing C is

$$t - (x + 2)(y + 2) = 0.$$

Example 5.4 (The non-powers of 2)**.** Let $S = \{0, 3, 5, 6, 7, 9, \ldots\} \subset \mathbb{N}$ denote the set of natural numbers that are not powers of 2. We find a Diophantine equation $D_S(t, x, y) = 0$ in one parameter t and two variables x, y such that the identity

$$S = \Big\{ t \in \mathbb{N} \colon\ (\exists x, y \in \mathbb{N}),\ D_S(t, x, y) = 0 \Big\}$$

holds. A natural number is not a power of 2 if and only if it has an odd divisor greater than or equal to 3. This means that we can write

$$S = \Big\{ t \in \mathbb{N} \colon\ (\exists x, y \in \mathbb{N}),\ t = (2x + 3)y \Big\},$$

since $(2x+3)$ ranges among all odd natural numbers that are at least equal to 3. Thus, a Diophantine equation describing the set S of non-powers of 2 is

$$t - (2x+3)y = 0.$$

In the two examples just presented (composite numbers and non-powers of 2), we found two simple Diophantine equations, each depending on one parameter and only two variables, describing them. In either case, the complement of each one of the sets is clearly listable. The set of powers of 2 can be listed by a program that starts by setting a parameter j to 1 and then loops through the following steps: print j, replace j by $2j$. Similarly, the set of primes is the set of natural numbers n that are not divisible by any natural number $d \in \{2, 3, \ldots, n-1\}$: this condition can be easily converted into a computer program to list the set of prime numbers.

As a consequence of the equivalence between listable and Diophantine sets, we obtain that the set of powers of 2 and the set of prime numbers are both Diophantine sets. In these two cases, finding a Diophantine equation describing the corresponding set is much more challenging than what we have seen so far.

An observation of Putnam makes the existence of such Diophantine equations even more striking. Let $D(t, x_1, \ldots, x_n) = 0$ be a Diophantine equation in one parameter t and any number n of variables. The Diophantine set S_D of values of $t \in \mathbb{N}$ for which the corresponding Diophantine equation admits a solution is also equal to the set of non-negative values of a polynomial. Indeed, let P_D denote the polynomial

$$P_D = (t+1)\big(1 - D(t, x_1, \ldots, x_n)^2\big) - 1.$$

The only chance for the polynomial P_D to obtain a non-negative value is to evaluate it at a $(1+n)$-tuple which is a root of D (recall that we are only allowed non-negative inputs). By definition, this means that t is in the set S_D and that the value of P_D is

$$P_D = (t+1)\big(1 - D(t, x_1, \ldots, x_n)^2\big) - 1 = (t+1) \cdot (1 - 0^2) - 1 = t.$$

This simple "trick" allows us to interchange freely between Diophantine subsets of $\mathbb{N}$ and non-negative values of polynomials with integer coefficients.

Before the equivalence of listable and Diophantine sets was proved, this observation was one of the main sources of disbelief at the proposed equivalence. For instance, we already stated that the set of prime numbers is listable. It follows that there is a polynomial with integer coefficients in several variables whose non-negative values are *precisely* the prime numbers. Nowadays, there are many explicit examples of such polynomials, but at the time, this was seen as very unlikely to be true. For instance, the formula available here is a polynomial with integer coefficients in 26 variables whose non-negative values are exactly the prime numbers. Without going into the details of how to construct such a polynomials, you can see that it is of the form $(k+2) \cdot (1 - (\text{sum of squares}))$, in line with Putnam's observation discussed above.

The work of many mathematicians culminated in a result of Davis, Putnam and Robinson about the structure of listable sets. Assuming that the set of triples $\{(a, b, a^b) \colon a, b \in \mathbb{N}\}$ is Diophantine, then every listable set is Diophantine. Even more, further work of Robinson showed that to prove the equivalenece of listable and Diophantine sets, it suffices to show that the set of powers of 2 is Diophantine. This last step was finally achieved by Matiyasevich.

6 Possible further topics

Existence of a "universal Diophantine equation"

The set of all Diophantine subsets of $\mathbb{N}$ is listable. This is simply because the set of Diophantine equations in a finite (but not fixed) number of variables is listable. Hence, we can fix an enumeration $D_0, D_1, \ldots, D_n, \ldots$ of all Diophantine subsets of $\mathbb{N}$. The "universal Diophantine set" is the subset

$$U = \{(n, t) \colon \quad n \in \mathbb{N},\ t \in D_n\}$$

of $\mathbb{N}^2$. The set U is Diophantine. As a consequence, there is a natural number m and a polynomial $P(t, x_1, \ldots, x_m)$ in $m+1$ variables with the following property. For every Diophantine set D contained in $\mathbb{N}$, there exists a natural number t_D, such that the set D is the set of non-negative values of $P(t_D, x_1, \ldots, x_m)$, as $x_1, \ldots, x_m$ range in $\mathbb{N}$. In this sense, the polynomial P "describes all Diophantine subsets of $\mathbb{N}$".

Gödel incompleteness theorem

Given a language, a listable set of axioms in this language and a listable set of logical deductions among statements in this language, we can program a computer to list all provable statements. This is similar to what we already saw. We form all possible deductions using an increasing number of axioms and an increasing number of deduction steps. The result, is that we can list all theorems. If the language is sufficiently complex so as to encode the standard arithmetic (e.g. the axioms are the axioms of ZFC), then the existence of undecidable Diophantine equations implies that there are true statements about the natural numbers that are not provable from the given list of axioms.

Universal Turing machines, cellular automata and Rule 110

Once the equivalence of Diophantine and listable sets is established, it is natural to wonder how simple can a computer be to still be able to generate *all* listable sets: what are the simplest universal Turing machines?

One of the simplest examples is called Rule 110. This "computer" reads 3 consecutive binary digits and returns a binary digit. The rules that it follows are given in Table 1.

Input	Output	Input	Output
111	0	011	1
110	1	010	1
101	1	001	1
100	0	000	0

Table 1: Instructions for Rule 110

The "program" is a string of 0s and 1s that the machine scans from left to right. The output is another string of 0s and 1s, constructed following the rules above.

Example 6.1. Suppose that we feed the string 00010111 to Rule 110. The machine starts reading the leftmost triple of digits 000 and writes 0, according to Table 1. Next, it reads the digits 001 in positions 2,3,4 and appends 1 to the output. It continues by reading 010 and appending 1. Continuing in this manner, the output of Rule 110 on input 00010111 is

011110.

Sample problems whose resolution is equivalent to the solubility of a Diophantine equation

Besides many problems in number theory that are directly asking for solubility of a Diophantine equation, there are problems that can be reduced to such a form. Examples are

- the existence of a non-trivial zero of the Riemann ζ-function with real part different from $\frac{1}{2}$ (cf. Riemann Hypothesis);
- the existence of an even number greater than or equal to 4 that is not a sum of two primes (cf. Goldbach's conjecture).