Hilbert's Tenth Problem

Recall that a Diophantine equation is an equation whose solutions are required to be be integers.

Hilbert's Tenth Problem (1900): "Given a Diophantine equation with any number of unknowns and with integer coefficients: devise a process, which could determine by a finite number of operations whether the equation is solvable in integers."

In this statement, the equation is required to be a polynomial. We find that

- such a process could be used to solve simultaneous equation systems

$$f_1(x_1, \dots x_n) = \dots = f_m(x_1, \dots x_n) = 0 \Leftrightarrow \sum_{i=1}^{n} f_i(x_1, \dots x_n)^2 = 0$$

- it would suffice to find an algorithm which solves single equations of degree 4, or systems with linear equations and equations of the form $(x_i = x_j^2)$
- solving over $\mathbb{Z}$ is equivalent to solving over the natural numbers $\mathbb{N} = \{0, 1, 2 \dots\}$

$$x \in \mathbb{N} \Leftrightarrow \exists x_1, x_2, x_3, x_4 \in \mathbb{Z} : x = x_1^2 + x_2^2 + x_3^2 + x_4^2$$

$$x \in \mathbb{Z} \Leftrightarrow \exists x_1, x_2 \in \mathbb{N} : x = x_1 - x_2$$

However, none of these are possible, as Hilbert's tenth problem is impossible. This was proved by Yuri Matiyasevich in 1970, building on work of Martin Davis, Julia Robinson, and Hilary Putnam.

Definition 1 *A set $S \in \mathbb{N}^n$ is Diophantine if, for a $m+n$-ary polynomial D,*

$$S = \{(a_1, \dots, a_n) | \exists x_1, \dots x_m \in \mathbb{N} : D(a_1, \dots a_n, x_1, \dots, x_m) = 0\}$$

The polynomial D, along with m and n, form a "Diophantine representation" of M.

Theorem 1 (Matiyasevich) *Every Turing-recognizable set is Diophantine.*

When this theorem is combined with the (much easier) fact that every Diophantine set is Turing-recognizable, we see that the notions of Diophantine and Turing-recognizable coincide. Matiyasevich's theorem also provides us with a computable method for translating a description of a Turing-recognizable set (Turing machine) into an equivalent Diophantine polynomial. The unsolvability of Hilbert's tenth problem then follows from the negative solution to the halting problem:

Theorem 2 (Turing, 1936) *There is no single algorithm which takes a Turing machine description as input, and outputs whether there exists an input on which that Turing machine halts.*

The idea is that, if we had a Diophantine decider $\mathcal{A}(D)$, we could construct a Turing decider $\mathcal{A}'(T) := \mathcal{A}(TuringMachineToDiophantineEquation(T))$; since the latter is impossible, the former is as well.

A Diophantine Language

We can define a language out of Diophantine equations, starting with the notion of Diophantine set (Definition 1). In general, we will assume that all variables are non-negative integer numbers, that is, elements of $\mathbb{N}$, since this simplifies some elements of the proof. However, the coefficients of the polynomials involved can still be negative.

We find that Diophantine sets in $\mathbb{N}^n$ are closed under union and intersection, with representations

$$D_1(a_1, \ldots, a_n, x_1, \ldots x_m) \cdot D_2(a_1, \ldots, a_n, x_1, \ldots x_{m'})$$

$$D_1(a_1, \ldots, a_n, x_1, \ldots x_m)^2 + D_2(a_1, \ldots, a_n, y_1, \ldots y_{m'})^2$$

respectively. Diophantine sets are also closed under projection.

A Diophantine relation is any relation $R(a_1, \ldots, a_m)$ such that the set of tuples satisfying the relation is Diophantine; we can show that $=, \neq, >, \geq$ are Diophantine. A relation of one variable is called a property, such as the Diophantine properties Even and SquareFree. The closure of Diophantine sets with respect to union and intersection means that if R_1, R_2 are Diophantine relations, then so are R_1 & R_2 and $R_1 \vee R_2$.

A Diophantine function is a function whose graph is Diophantine, namely $f : \mathbb{N}^n \to \mathbb{N}$ is Diophantine if and only if there is a polynomial D such that

$$a = f(b_1, b_2, \ldots, b_n) \Leftrightarrow \exists x_1, \ldots, x_m : D(a, b_1, \ldots, b_n, x_1, \ldots, x_m) = 0$$

To keep things conceptually simple, we can define Diophantine sets, relations, and functions out of other Diophantine objects. For example, from the Diophantine divisibility relation

$$a|b \Leftrightarrow \exists x : ax - b = 0$$

we can build the remainder function

$$a = \text{rem}(b, c) \Leftrightarrow a < c \ \& \ c|b - a.$$

Similarly, we show that the functions div, lcm, and gcd are Diophantine.

Exponentiation is Diophantine

The next function we need to show is Diophantine is exponentiation. In other words, we want to show that the set

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

has a Diophantine representation. I will give only some of the details about how this is accomplished. A 1961 paper by Robinson, Davis and Putnam contained a proof that, if exponentiation is Diophantine, then Hilbert's tenth problem is unsolvable (although they used a different approach than what is described here). Chronologically, exponentiation was the last step in the proof, and this is what Matiyasevich showed in 1970. For more details on the approach I am describing, see Hilbert's Tenth Problem by Matiyasevich (1993).

Encoding Tuples as Integers

Any particular Diophantine equation has a fixed number of unknowns. However, if we can figure out a way to represent a variable-length list through a fixed number of variables (namely, three), we can effectively have a single equation represent an arbitrarily complicated computation.

We say that the triple (a, b, n) is a code for the sequence $(a_1, \ldots, a_n)$ if

$$a = a_n b^{n-1} + a_{n-1} b^{n-2} + ... + a_1 b^0 \qquad \text{and} \qquad b > a_i, i = 1, ..., n.$$

This type of coding is quite versatile, once we have proved that exponentiation is Diophantine. We can determine whether a particular tuple represents a code,

$$\text{Code}(a, b, c) \Leftrightarrow b \geq 2 \ \& \ a < b^c$$

and also extract individual elements (the dth element) from a code (a, b, c)

$$\text{Element}(a, b, d) \Leftrightarrow \exists xyz[d = z + 1 \ \& \ a = xb^d + eb^x + y \ \& \ e < b \ \& \ y < b^z]$$

Likewise, if we have two codes with the same base b, then we can add, subtract, and append tuples together with little difficulty.

Note that in this system, $((b + 1)^n, b, n + 1)$ is an encoding of the tuple

$$\left(\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}\right)$$

if b is large compared to n (say, $b = 2^n + 1$). Thus

$$c = \binom{n}{m} \Leftrightarrow c = \text{Element}((2^n + 2)^n, 2^n + 1, m + 1).$$

From this Diophantine representation of Pascal coefficients, we can get a Diophantine representation of factorial. Note that

$$m! = \lim_{n \to \infty} \frac{n^m}{\binom{n}{m}}$$

and although there is no way to express a limit in general Diophantine terms, the limit becomes close enough that

$$m! = n^m \operatorname{div} \binom{n}{m} \quad \text{for } n \geq (m + 1)^{m+2}$$

and this allows us to express the factorial function as Diophantine. This in turn allows us to show that the set of primes is Diophantine:

$$\text{Prime}(a) \Leftrightarrow a \geq 2 \ \& \ \gcd(a, (a - 1)!) = 1$$

Tuple Equality and Tuple Maps

Next, we can use Kummer's Theorem (1852) to test when one tuple has elements greater than the corresponding elements of another tuple, when their common base b is prime.

$$\begin{aligned}\text{PNotGreater}(a, a', b) &:= \text{Prime}(b)\ \&\ a_i \le a'_i, i = 1, \ldots, n \\ &\Leftrightarrow \text{Prime}(b)\ \&\ b \nmid \binom{a'}{a}.\end{aligned}$$

Amazingly, this lets us construct a way to determine if two given tuples are equal, even if they have different bases. Suppose we have two codes in different bases b, b', with b' prime and $b' > b$. Clearly, a necessary condition for (a, b, n) and (a', b', n) to represent the same code is that each element of the n-tuple (a', b', n) must be smaller than b, or

$$\text{PNotGreater}(a', \text{Repeat}(b{-}1, n, b'), b') \text{ where } \text{Repeat}(x, n, b') = x(b'^n{-}1)/(b'{-}1)$$

We see that if the codes (a, b, n) and (a', b', n) represent the same tuple, then in addition to the smallness condition,

$$a \equiv a' \pmod{b' - b} \qquad \text{and} \qquad a < b^n.$$

Furthermore, if $b^n < b' - b$, then these three conditions cause a', b', b to uniquely determine a, and we can conclude that

$$\begin{aligned}\text{PEqual}(a', b', a, b, n) &:= \text{Prime}(b)\ \&\ b' > b + b^n\ \&\ (b' - b) | (a - a')\ \&\ a < b^n \\ &\quad \&\ \text{PNotGreater}(a', \text{Repeat}(b - 1, n, b'), b') \\ &\Leftrightarrow \text{Prime}(b)\ \&\ b' > b + b^n\ \&\ a_i = a'_i, i = 1, \ldots, n\end{aligned}$$

and from PEqual we construct the Diophantine relations Equal and NotGreater.

The last construction we need to simulate Turing machines is a way of extending functions to work on whole tuples of a fixed base b. Namely, if $f : \{0, 1, \ldots, b - 1\} \Rightarrow \mathbb{N}$ is Diophantine, then we can show that

$$f[b](a, n) : (a_1, \ldots, a_n) \mapsto (f(a_1), \ldots, f(a_n))$$

is also Diophantine. It would suffice to extract the b characteristic tuples $\{(h_{i,1}, \ldots, h_{i,n})\}_{i=0}^{b-1}$ where $h_{i,k} = 1$ if $a_i = k$ and 0 otherwise, and return

$$\sum_{i=0}^{b-1} f(i) \cdot h_i.$$

To extract these tuples, we use the $b + b(b - 1)/2 + 1$ conditions

$$\begin{gathered}\text{NotGreater}(h_i, \text{Repeat}(1, n, b), b), \forall i = 0, \ldots, b - 1 \\ \text{NotGreater}(h_i + h_j, \text{Repeat}(1, n, b), b), \forall i, j = 0, \ldots, b - 1 : i < j \\ \textstyle\sum_{i=0}^{b-1} i \cdot h_i = a.\end{gathered}$$

Finally, we prove Matiyasevich's Theorem using a method similar to the Cook-Levin theorem. Our construction will map a Turing machine T to a Diophantine equation $D(T)$ such that $D(T)$ has a solution iff T accepts some input.