Hilbert's Tenth Problem

by
Andrew J. Misner

A project submitted to the Department of
Mathematical Sciences in conformity with the requirements
for Math 4301 (Honour's Seminar)

Lakehead University
Thunder Bay, Ontario, Canada

Abstract

In 1900, David Hilbert presented a problem at the International Congress of Mathematics questioning the solvability of Diophantine Equations within the integers. If an algorithm can be found to show that a specific Diophantine equation can be solved within the integers, we can prove Hilbert's problem. Using the idea of a Turing machine, as well as contributions from Julia Robinson, Martin Davis, Hilary Putnam, and Yuri Matiyasevich, we will show that no such algorithm exists, proving Hilbert's problem is unsolvable.

Acknowledgements

I would like to thank my supervisor, Dr. Andrew J. Dean, for his guidance and assistance in the completion of this paper. I would also like to thank Dr. Adam Van Tuyl for monitoring my pace of delivery, as well as his very helpful LaTeX tutorials.

Contents

CHAPTER 1

A Brief Introduction

In 1900, David Hilbert gave an address at the International Congress of Mathematicians in which he gave a list of 23 problems that would influence the world of mathematics. In this project, we will focus on his tenth problem. In his tenth problem, Hilbert talks about Diophantine equations, and if these equations can be computed using any form of algorithm. Without proper resources to tackle this problem, no work began on this problem until the work of Martin Davis. The problem was completed by Yuri Matiyasevich in 1970. The invention of the Turing Machine in 1936 was crucial to form a solution to this problem.[**1**]

PROBLEM 1.1. **(Hilbert's Tenth Problem)**[**3**] *Given a Diophantine equation: To devise an algorithm according to which it can be determined in a finite number of operations whether the equation is solvable in the integers.*

As it turns out, there is no solution to Hilbert's Tenth Problem, thus making the problem unsolvable. In Hilbert's 1900 address, he gives the following definition of an unsolvable problem:

"Occasionally it happens that we seek the solution under insufficient hypotheses or in an incorrect sense, and for this reason do not succeed. The problem then arises: to show the impossibilty of the solution under the given hypotheses, or in the sense contemplated." (Hilbert, 1900) [**6**]

Thus, if we are able to show that Hilbert's Tenth Problem is impossible, it suffices that it is unsolvable.

1. Introduction to Diophantine Equations

1.1. Definitions and Examples. In 1900, the concept of a polynomial was widely known. It was extensively studied by mathematicians, but Hilbert was interested in one family of polynomials, the Diophantine polynomials. Recall the definition of a polynomial:

DEFINITION 1.2. A *polynomial* is an expression of finite length, made up of variables, coefficients contained within a ring of coefficients, and non-negative integer exponents.

EXAMPLE 1.3. $5x^2 - 3.2x + 7$ is an example of a polynomial, on the other hand, $3x^2 - 3/x + x^{1/2}$ is not a polynomial since there are negative and non-integer exponents.

Though there were limitations on the exponents, Diophantine equations further limited the definition by placing restrictions on the coefficients.

DEFINITION 1.4. [**4**] A *Diophantine equation* is a polynomial of the form

$$P(X_1, \ldots, X_n) = 0,$$

where all coefficients are contained within the ring of integers $\mathbb{Z}$.

EXAMPLE 1.5. $5x^2 - 3.2x + 7$ is a polynomial, but not a Diophantine equation, since 3.2 is not an integer. However, $5x^2 - 3x + 7$ is a Diophantine equation.

DEFINITION 1.6. [**4**] A *Diophantine set*, denoted $S \subseteq \mathbb{N}^n$, is a set of ordered n-tuples where $(X_1, \ldots, X_n) \in S$ if there exists positive integers $Y_1, \ldots, Y_m$ such that the Diophantine equation is satisfied, i.e. $P(X_1, \ldots, X_n, Y_1, \ldots, Y_m) = 0$.

We define $\mathbb{N}$ as the natural numbers not including zero.

EXAMPLE 1.7. [**2**] The composite numbers are an example of a Diophantine set. Since any composite number can be written as a product of two positive integers greater than 1, we can set this up as a Diophantine set:

$$x \in S \Leftrightarrow (\exists y, z \in \mathbb{N})[x = (y + 1)(z + 1)]$$

DEFINITION 1.8. [**4**] A *Diophantine function* f is a function of n arguments $X_1, \ldots, X_n$ such that $\{\langle X_1, \ldots, X_n, y\rangle | y = f(X_1, \ldots, X_n)\}$ is a Diophantine set.

EXAMPLE 1.9. [**2**] We can form three important Diophantine functions using the idea of triangular numbers. A triangular number is of the form $T(n) = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$. Now as n increases, so does the function $T(n)$. Thus, for every $z \in \mathbb{N}$, there is an $n \geq 0$ such that

$$T(n) < z \leq T(n + 1) = T(n) + n + 1.$$

Now we are able to write z as:

$$z = T(n) + y; \qquad y \leq n + 1.$$

We can also find an x such that we can uniquely write z as:

$$z = T(x + y - 2) + y.$$

Now we can define three functions $L(z) = x, R(z) = y, P(x, y) = z$ and show that they are Diophantine functions, written as:

$$z = P(x, y) \Leftrightarrow 2z = (x + y - 2)(x + y - 1) + 2y \tag{1.1}$$

$$x = L(z) \Leftrightarrow (\exists y)[2z = (x + y - 2)(x + y - 1) + 2y] \tag{1.2}$$

$$y = R(z) \Leftrightarrow (\exists x)[2z = (x + y - 2)(x + y - 1) + 2y]. \tag{1.3}$$

THEOREM 1.10. **(Pairing Function Theorem)**[**2**]
Given the Diophantine functions $P(x, y), L(z)$, and $R(z)$, we have the following properties:
(1)$\forall x, y, L(P(x, y)) = x, R(P(x, y)) = y$
(2)$\forall z, P(L(z), R(z)) = z, L(z) \leq z, R(z) \leq z$

PROOF. The proof of Theorem 1.9 is straightforward. By definition, $L(z) = x, R(z) = y, P(x, y) = z$. Thus the properties are obviously correct. □

DEFINITION 1.11. Two integers r and s are *relatively prime* if gcd(r,s)=1.

THEOREM 1.12. *If a and b are relatively prime with $b > 0$, then there exists an integer $\tilde{a}$ such that $\tilde{a}$ is the inverse of a modulo b. That is, $\tilde{a}a \equiv 1 \pmod{b}$.*

PROOF. Since a and b are relatively prime, we know gcd(a, b)=1. Therefore, we can find integers r and s such that

$$ra + sb = 1.$$

We now know that

$$ra + sb \equiv 1 \pmod{b}.$$

Since sb is divisible by b, we know $sb = 0 \pmod{b}$. Therefore, we can see that

$$ra = 1 \pmod{b}$$

Thus, we have found an inverse $\tilde{a} = r$. □

THEOREM 1.13. **(Chinese Remainder Theorem)**[**5**]
Let $a_1, \ldots, a_n$ be any positive integers. Let $m_1, \ldots, m_n$ be a group of pairwise relatively prime moduli. Then there exists an integer x such that

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_n \pmod{m_n} \end{aligned}$$

and x is unique modulo $m = m_1 m_2 \cdots m_n$.

PROOF. To begin the proof, we will define a group of integers $M_1, \ldots, M_n$ such that

$$M_i = m/m_i$$

That is, M_i is the product of all moduli except m_i. Since $m_1, \ldots, m_n$ are relatively prime, we can see that gcd(m_i, M_i) = 1. By Theorem 1.11, we can find an integer z_i such that z_i is an inverse of M_i modulo m_i. That is,

$$M_i z_i \equiv 1 \pmod{m_i}.$$

Now we are able to form the sum

$$x = a_1 M_1 z_1 + \cdots + a_n M_n z_n.$$

Now if we have an integer $j \neq i$, we have $M_j \equiv 0 \pmod{m_i}$. From above, we have $M_i z_i \equiv 1 \pmod{m_i}$, multiplying both sides by a_i, we have

$$x \equiv a_i M_i z_i \equiv a_i \pmod{m_i}$$

for all $i = 1, 2, \ldots, n$. Thus, x is a solution to the n congruences.

Now we must show that x is a unique solution. Suppose we have a second solution to the system equations, y. That is,

$$y \equiv x \equiv a_i \pmod{m_i}$$

for all $i = 1,2,\ldots,n$. So each m_i divides $y - x$. So we have

$$y \equiv x \pmod{m_i}$$

for all i. Now y can be written as $y = x + k \cdot m_i$ for some integer k. Since all m_i are relatively prime, we must have $k = 0$. Thus $y = x$ and x is a unique solution to the system of equations.

□

Now we may define a Diophantine function $S(i, u)$ such that $S(i, u) = w$, where w is a positive integer such that:

$$w \equiv L(u) \pmod{(1 + iR(u))}$$

THEOREM 1.14. **(Sequence Number Theorem)**[**2**]
There is a Diophantine function $S(i, u)$ such that
(1) $S(i, u) \leq u$
(2) for each sequence $a_1, \ldots, a_n$, there is a number u such that

$$S(i, u) = a_i \ for \ 1 \leq i \leq n$$

PROOF. The proof of (1) is straightforward. Since $S(i, u)$ is the remainder of $L(u)$ divided by $1 + R(u)$, we can see that $S(i, u) \leq L(u)$. Using part 2 of Theorem 1.9, we know that $L(u) \leq u$. Thus, $S(i, u) \leq u$. The proof of (2) will be omitted, as it is more complicated, requiring variable manipulation and use of the Chinese Remainder Theorem. □

Now that we have defined Diophantine equations, sets, and functions, we are able to begin work on the problem at hand, Hilbert's Tenth Problem. The rest of the paper will be constructed as follows: In Chapter 2, a few concepts in computer science will be introduced, and the Halting Problem with be proven, giving a basis for proving Hilbert's Tenth Problem. In Chapter 3, advanced topics in Diophantine theory will be introduced and the foundation for the final proof will be formed. In Chapter 4, Hilbert's Tenth Problem will be proven to be unsolvable. In Chapter 5, I will explain my reasoning behind taking on this project, as well as present ideas for future study.

CHAPTER 2

Computability and Unsolvability

1. Introduction to the Age of Computers

Alan Turing is known as the father of computer science. He was a mathemician, logician, and a cryptanalyst. His most prominent work was in the field of computer science. He coined such terms as algorithm and computation, and is well known for the invention of the Turing machine in 1936, an early form of a computation device.

1.1. Decision Problems and the Entscheidungsproblem.

Definition 2.1. A *Decision Problem* is a problem in which an algorithm is used to find whether or not a function has a specific property P. The algorithm will output either yes or no.

1.2. The Work of Kurt Gödel. [**2**] Kurt Gödel was an Austrian American logician and mathematician. He is most famous for his two incompleteness theorems, and the proofs of these theorems gave Martin Davis the ability to begin work on Hilbert's Tenth Problem.

Kurt Gödel was one the the many mathematicians involved in the work of logic. Also being interested in computer science, he spent time working on a logical decision problem, the Entscheidungsproblem.

Definition 2.2. **(Hilbert, 1928)** An *Entscheidungsproblem* is a problem in which a logical statement is taken by an algorithm, and the algorithm will output if the statement is true or false.

Definition 2.3. An *axiomatic system* is any set of axioms, when used together, can logically derive a theorem.

Definition 2.4. A *consistent* theorem is any theorem that does not contain a contradiction.

Definition 2.5. A *contradiction* in logic occurs when two declarative statements, when taken together, yield opposite results.

Definition 2.6. An axiomatic system is called *complete* if for each statement in the system, there is a method of deriving that statement.

THEOREM 2.7. **(Gödel's First Incompleteness Theorem)**
If a computable axiomatic system is consistent, it cannot be complete. Thus, more axioms must be added to the system to make it complete.

THEOREM 2.8. **(Gödel's Second Incompleteness Theorem)**
The consistency of the axioms cannot be proven within the axiomatic system.

In order for Gödel to prove these theorems in mathematics, he had to form a method of encoding mathematical statements into natural numbers. He called this method as **Gödel numbering**.

DEFINITION 2.9. *Gödel numbering* is the process of using a function to turn letters and mathematical symbols into natural numbers.

The easiest and most well-known method of this is the ASCII coding system.

EXAMPLE 2.10. Using ASCII Decimal Language, we can easily encode any statement, such as the transitive property

If A=B and B=C, then A=C.
This translates to 073-102-032-065-061-066-097-109-100-032-066-061-067-044-032-116-104-101-110-032-065-032-067-046
This gives the Gödel number:
073102032065061066097109100032066061067044032116104101110032065032067046

Gödel numbering is important as it gives a code that a Turing machine is able to read.

2. Computability

DEFINITION 2.11. A *Turing Machine* is an abstract computing machine that can use a predefined set of rules to determine a result from a set of input variables.

A Turing machine is made up of three main parts: the tape, the head, and the table. The *tape* is a long strip of information that can be read by the Turing machine. This strip is code for the type of algorithm that the Turing machine will perform. The *head* of the machine reads each piece of the tape, and moves the tape left or right depending on the state of the problem or from what is read on the tape. The *table* is the computational part of the Turing machine. It holds the information that the head has read, and keeps track of the state of the problem. The *state* of the problem is what part of the algorithm the machine is at. The state determines where the head should move the tape, and also what the table should compute next.

DEFINITION 2.12. A *computable* function is one that is able to be computed by a program or computing machine in a finite amount of time. The set of computable functions is the same as the set of recursive functions.

2.1. Unsolvability and the Halting Problem. As people began solving problems with the use of a Turing machine, it was pondered if there existed problems that were unable to be solved using a form of computation. One of the first unsolvable problems was the Halting Problem. it is known as one of the most highly regarded problems in computer science as it gives a basis for the formation of many other unsolvable problems.

PROBLEM 2.13. **(Halting Problem, 1936)** *Is there a procedure that can take a program and some string data value as input, and determine whether or not the program will halt at that input value.*

THEOREM 2.14. **(Turing, 1936)**[**2**] *There exists no such procedure that satisfies the halting problem.*

PROOF. Assume that there exists a solution to the halting problem. Call this procedure $H(P, I)$ where P is the program and I is the string input. If P halts on I, then $H(P, I)$ will generate the string "halt" as output. If P does not halt on I, H will generate the string "loops forever" as output.

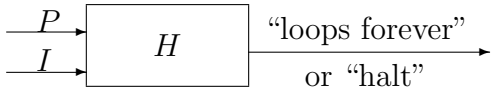

FIGURE 1. Procedure H

Since a program is just a sequence of characters, the program itself can be considered a string, and can thus be used as the input value I. Now suppose that we create a second procedure $K(P)$ such that the program reads the output of $H(P, P)$ and does the opposite. Hence, if the output of $H(P, P)$ is "loops forever", $K(P)$ will output "halt". Similarly, if $H(P, P)$ outputs "halt", $K(P)$ will output "loops forever".

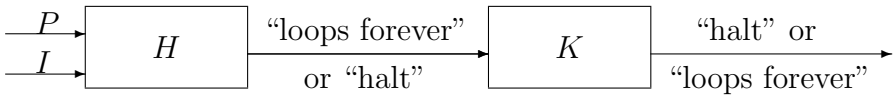

FIGURE 2. Procedure K

Now suppose that we begin our procedure H with input $H(K, K)$. If $H(K, K)$ outputs "halt", then $K(K)$ will output "loops forever". Similarly, if $H(K, K)$ outputs "loops forever", then $K(K)$ will output "halt". Therefore, we arrive at a contradiction for both cases.

Thus, we cannot can a procedure H that will always give correct answers. Therefore, there exists no algorithm that can solve the halting problem.

□

With the creation of an unsolvability proof, mathematicians were able to use the basis of the halting problem proof to prove the unsolvability of Hilbert's Tenth Problem.

CHAPTER 3

Setting up our Unsolvability Proof

There are many different ways to define the set of recursive functions, one way is:

DEFINITION 3.1. A function is *recursive* if it can be formed using the following initial functions

$$c(x) = 1,$$
$$s(x) = x + 1,$$
$$U_i^n(x_1, \ldots, x_n) = x_i \qquad 1 \leq i \leq n,$$
$$S(i, u),$$

using a combination of three operations:

Composition is the operation of combining functions f of m variables with g of n variables:

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$$

Primitive Recursion is the operation of recursively solving for a function $h(x_1, \ldots, x_n, z)$ using functions f and g:

$$h(x_1, \ldots, x_n, 1) = f(x_1, \ldots, x_n)$$

$$h(x_1, \ldots, x_n, t+1) = g(t, h(x_1, \ldots, x_n, t), x_1, \ldots, x_n)$$

Finally, *Minimalization* is the operation of minimizing y such that y satisfies $f(x_1, \ldots, x_n, y) = g(x_1, \ldots, x_n, y)$

$$h(x_1, \ldots, x_n) = min_y[f(x_1, \ldots, x_n, y) = g(x_1, \ldots, x_n, y)]$$

In order to prove that Hilbert's Tenth Problem is unsolvable, we must prove several other things first. The first type of equation tested to be Diophantine was the exponential (EXP) function, proposed by Julia Robinson.

THEOREM 3.2. [**2**] *The exponential function $h(n, k) = n^k$ is Diophantine.*

The proof of this theorem is a basic exercise in number theory. Due to its length, we will omit it from this paper. A complete proof can be found in [**2**]

We may now introduce the topic of Diophantine predicates. We can combine Diophantine functions with other Diophantine functions using the & (and), $\vee$ (or), and $\exists$ (exists) operators, thus forming other Diophantine functions.

EXAMPLE 3.3. Since we know that the exponential function is Diophantine, using Diophantine predicates, we can show that $h(u, v, w) = u^{v^w}$ is Diophantine. We can write h in the form:

$$y = u^{v^w} \Leftrightarrow (\exists z)(y = u^z \ \& \ z = v^w)$$

Since y and z are both Diophantine functions, combining them together using the & operator will form another Diophantine function h.

The next step in proving Hilbert's Tenth problem is unsolvable is to show that every Diophantine function is a recursive function.

THEOREM 3.4. [**2**] *A function is Diophantine if and only if it is recursive.*

PROOF. To prove that all Diophantine functions are recursive, we must show that any Diophantine function can be expressed in terms of the initial functions from Definition 2.2. Hence, we shall break this proof into cases:

(1) $x + y$ is recursive because:

$$x + 1 = s(x),$$
$$x + (t + 1) = s(x + t) = g(t, x + t, x),$$

where $y = t + 1$ and $g(u, v, w) = s(U_2^3(u, v, w))$.

(2) $x \cdot y$ is recursive because:

$$x \cdot 1 = U_1^1(x)$$
$$x \cdot (t + 1) = (x \cdot t) + x = g(t, x \cdot t, x),$$

where $y = t + 1$ and $g(u, v, w) = U_2^3(u, v, w) + U_3^3(u, v, w)$.

(3)The constant function $c_k(x) = k$ is recursive because:

$$c_1(x) = c(x)$$
$$c_{k+1}(x) = c_k(x) + c(x).$$

Now the proof is straightforward. Since we can write any Diophantine function as a composition of these three recursive functions, any Diophantine function must be recursive.

Now we must prove the converse, that is, all recursive functions are Diophantine. From Theorem 1.14, we know that $S(i, u)$ is a Diophantine function. By definition, we also know that $c(x)$, $s(x)$, and $U_i^n(x_1, \ldots, x_n)$ are Diophantine functions. So it suffices to show that Diophantine functions are closed under composition, primitive recursion, and minimalization.

(1) *Composition*: If $h(x_1,\ldots,x_n) = f(g_1(x_1,\ldots,x_n),\ldots,g_m(x_1,\ldots,x_n))$, and $f, g_1,\ldots,g_m$ are Diophantine functions, we can write h as a Diophantine function as follows:

$$y = h(x_1,\ldots,x_n) \Leftrightarrow (\exists t_1,\ldots,t_m)[t_1 = g_1(x_1,\ldots,x_n)\ \&\ldots\&\ t_m = g_m(x_1,\ldots,x_n)\ \&\ y = f(t_1,\ldots,t_m)$$

(2) *Primitive Recursion*: If

$$h(x_1,\ldots,x_n,1) = f(x_1,\ldots,x_n)$$
$$h(x_1,\ldots,x_n,t+1) = g(t,h(x_1,\ldots,x_n,t),x_1,\ldots,x_n),$$

and f and g are Diophantine, then using the Sequence Number Theorem, we can write $h(x_1,\ldots,x_n,1), h(x_1,\ldots,x_n,2),\ldots,h(x_1,\ldots,x_n,z)$ as follows:

$$y = h(x_1,\ldots,x_n,z) \Leftrightarrow (\exists u)\{(\exists v)[v = S(1,u)\ \&\ v = f(x_1,\ldots,x_n)]\ \&\ (\forall t)_{\le z}[(t = z)\vee(\exists v)(v = S(t+1,u)\ \&\ v = g(t,S(t,u),x_1,\ldots,x_n))]\ \&\ y = S(z,u)\}$$

(3) *Minimalization*: If

$$h(x_1,\ldots,x_n) = min_y[f(x_1,\ldots,x_n,y) = g(x_1,\ldots,x_n,y)]$$

where f and g are Diophantine, then h is Diophantine as follows:

$$y = h(x_1,\ldots,x_n) \Leftrightarrow (\exists z)[z = f(x_1,\ldots,x_n,y)\ \&\ z = g(x_1,\ldots,x_n,y)]\ \&\ (\forall t)_{\le y}[(t = y)\vee (\exists u,v)(u = f(x_1,\ldots,x_n,t)\ \&\ v = g(x_1,\ldots,x_n,t)\ \&\ (u < v\ \vee\ v < u)].$$

□

Example 3.5. The Diophantine function $8xyz^2 + 7x^2y + 12y^3z$ is recursive because it can be written as:

$$8xyz^2 + 7x^2y + 12y^3z = c_8(x)\cdot x\cdot y\cdot z\cdot z + c_7(x)\cdot x\cdot x\cdot y + c_{12}(x)\cdot y\cdot y\cdot y\cdot z$$

Definition 3.6. A set S of n-tuples of positive integers is called recursively enumerable if there are recursive functions $f(x,x_1,\ldots,x_n), g(x,x_1,\ldots,x_n)$ such that

$$S = \{\langle x_1,\ldots,x_n\rangle | (\exists x)[f(x,x_1,\ldots,x_n) = g(x,x_1,\ldots,x_n)]\}.$$

Theorem 3.7. [**2**] *A set S is Diophantine if and only if it is recursively enumerable.*

Proof. If S is Diophantine, we can find Diophantine equations P, Q such that :

$$\langle x_1,\ldots,x_n\rangle \in S \Leftrightarrow (\exists y_1,\ldots,y_m)[P(x_1,\ldots,x_n,y_1,\ldots,y_m) = Q(x_1,\ldots,x_n,y_1,\ldots,y_m)]$$
$$\Leftrightarrow (\exists u)[P(x_1,\ldots,x_n,S(1,u),\ldots,S(m,u)) = Q(x_1,\ldots,x_n,S(1,u),\ldots,S(m,u))]$$

so that S is recursively enumerable.

To prove the converse, assume S is recursively enumerable. That is, we have recursive functions $f(x,x_1,\ldots,x_n), g(x,x_1,\ldots,x_n)$ such that

$$\langle x_1,\ldots,x_n\rangle \in S \Leftrightarrow (\exists x)[f(x,x_1,\ldots,x_n) = g(x,x_1,\ldots,x_n)]$$
$$\Leftrightarrow (\exists x,z)[z = f(x,x_1,\ldots,x_n\ \&\ z = g(x,x_1,\ldots,x_n)].$$

From Theorem 3.4, S is Diophantine. □

The final idea we must introduce is a universal Diophantine set.

DEFINITION 3.8. [**2**] Given every known Diophantine function $D_1, D_2, D_3, \ldots$ we can form the *universal Diophantine set*:

$$\{\langle n, x\rangle | x \in D_n\}$$

Using Diophantine predicates on every Diophantine function, we can easily see that the universal Diophantine set is Diophantine.

CHAPTER 4

Hilbert's Tenth Problem is Unsolvable

Using the idea of a universal Diophantine set, we can create a set V such that

$$V = \{n | n \notin D_n\}$$

Since V contain no elements contained in any Diophantine set, we have the following theorem.

THEOREM 4.1. *V is not Diophantine.*

PROOF. This proof is handled exactly the same as the proof of the halting problem. Suppose V is Diophantine, in other words, $V = D_i$ for some fixed integer i. Now by testing to see if $i \in V$ we arrive at the following conclusions:

$$i \in V \Leftrightarrow i \in D_i \qquad i \in V \Leftrightarrow i \notin D_i$$

Since these conclusion contradict each other, V must not be Diophantine. □

THEOREM 4.2. *Define a function $g(n, x)$ by:*

$$g(n, x) = 1 \ \ if \ \ x \notin D_n$$
$$g(n, x) = 2 \ \ if \ \ x \in D_n$$

Then $g(n, x)$ is not recursive.

PROOF. [**2**] Suppose g is recursive. By Theorem 3.3, we know that g must be Diophantine. Therefore, we can write out g as a Diophantine function, say:

$$y = g(n, x) \Leftrightarrow (\exists y_1, \ldots, y_m)[P(n, x, y, y_1, \ldots, y_m) = 0].$$

Now from the definition of V above, we can write V as:

$$V = \{x | (\exists y_1, \ldots, y_m)[P(x, x, 1, y_1, \ldots, y_m) = 0]\}$$

Now we have a problem, as this contradicts Theorem 4.1. Using Definition 3.5, we can write:

$$x \in D_n \Leftrightarrow (\exists z_1, \ldots, z_k)[P(n, x, z_1, \ldots, z_k) = 0]$$

for some polynomial P.

Now if we were able to find an algorithm such that we could test if $P(n, x, z_1, \ldots, z_k) = 0$ had positive integer solutions, we could use the algorithm to test if any Diophantine equation has a solution in the integers. Then this algorithm would give us a solution to Hilbert's Tenth Problem! If such an algorithm existed, we would be able to test if $x \in D_n$, thus able to compute $g(n, x)$. In order for $g(n, x)$ to be computable, by Definition 2.2,

$g(n, x)$ would have to be recursive. Thus, we have arrived at a contradiction, proving $g(n, x)$ is not recursive, but also proving: □

THEOREM 4.3. *Hilbert's Tenth Problem is unsolvable.*

1. Conclusion

When David Hilbert presented his list of 23 problems, it was not his intention that the problems themselves were to be the main focus, but to focus on expanding the general knowledge of these mathematical topics. Even though Hilbert's Tenth Problem is unsolvable, attempting to solve the problem resulted in great advancements in the study of computer science and the study of Diophantine theory.

CHAPTER 5

Author's notes

When I began my journey into the study of mathematics, one of the main problems that piqued my interest was unsolved problems. The fact that there still exist questions in math that are left unanswered is something that intrigues me. When it came time for me to choose a topic of study, it was immediately clear that I wanted to solve a problem that was pondered for several decades. Since I minor in computer science, finding a problem that could incorporate computation was another reason for choosing Hilbert's Tenth Problem.

In future research, I would like to steer away from problems that have been determined to be solvable or unsolvable, and instead begin work on problems that do not yet have a solution. As graph theory is a relative topic to computer science, I would like to study unsolved problems in graph theory. Possible candidates include the Hadwiger-Nelson problem or the total coloring conjecture.

Bibliography

[1] American Mathematical Society. (1976). Mathematical Developments Arising from Hilbert Problems. Providence, Rhode Island: American Mathematical Society.

[2] Davis, M. (1958). Computability and Unsolvability. York, Pennsylvania: The Maple Press Company.

[3] Matiyasevich, Y. V. (1996). Hilbert's Tenth Problem. Cambridge, Massachusetts: Nauka Publishers.

[4] Mordell, L. J. (1969). Diophantine Equations. Cambridge, England: Academic Press.

[5] Rosen, K. E. (2007). Discrete Mathematics and Its Applications. Sixth Edition. New York, New York: McGraw Hill Publishing.

[6] Joyce, D. E. (1997). David Hilbert's 1900 Address. Worcester, Massachusetts: Clark University, available at http://aleph0.clarku.edu/ djoyce/hilbert/problems.html 1

2, 4, 5, 7, 9, 10, 11, 12, 13

1

2

3

1