

# Evaluating the Complexity of Mathematical Problems: Part 1

**Cristian S. Calude**

*University of Auckland, New Zealand*  
*www.cs.auckland.ac.nz/~cristian*

**Elena Calude**

*Massey University at Albany, New Zealand*  
*www.massey.ac.nz/~ecalude*

---

In this paper we provide a computational method for evaluating the complexity of a large class of mathematical problems in a uniform way. The method, which is inspired by *A New Kind of Science* [1], is based on the possibility of completely describing complex mathematical problems, such as the Riemann hypothesis, in terms of very simple programs. The method is illustrated on a variety of examples from different areas of mathematics and its power and limits are studied.

---

## 1. Introduction

---

Evaluating, or even guessing, the degree of difficulty of an open problem, or that of a solved problem before seeing its solution, is notoriously hard not only for beginners, but also for the most experienced mathematicians.

Can a uniform method be developed for evaluating in some objective way the difficulty of a mathematical problem? The question is not trivial because mathematical problems are very diverse. The Mathematics Subject Classification (MSC2000) based on the Mathematical Reviews and Zentralblatt MATH databases, contains over 5,000 two-, three-, and five-digit classifications [2]. Additionally, there is no clear indication that all, most, or even a large part of mathematical problems have any kind of “commonality” allowing a uniform evaluation of their complexity. How could one compare a problem in number theory with a problem in complex analysis or algebraic topology?

Surprisingly enough, such a commonality exists for many mathematical problems. One of them, the so-called *halting problem* [3], as discussed in this paper is based on the possibility of expressing a problem in terms of very simple programs that are reducible to a natural question in theoretical computer science. As a consequence, a uniform approach for evaluating the complexity of a large class of mathematical problems was developed.

This paper is structured as follows. In Section 2 a series of interesting mathematical problems are presented and analyzed in order to find their commonality. They are the infinity of primes, Goldbach's conjecture and other problems in number theory, the pigeonhole principle, Hilbert's tenth problem, the four color theorem, the Riemann hypothesis, and the Collatz and palindrome conjectures. We show how all these problems are closely related to the halting problem. Section 3 discusses the most (in)famous problem in theoretical computer science, the halting problem. Section 4 presents the method for evaluating complexity. Section 5 gives an example of a class of problems to which the proposed method applies; namely, the class of finitely refutable problems, and Section 6 examines the power and limits of the proposed method. We finish the paper with a few concluding remarks.

## 2. Some Interesting Mathematical Problems: What Do They Have in Common?

In this section we discuss some interesting, solved or open, mathematical problems searching for their possible "common computational structure".

### 2.1 The Infinity of Primes

Euclid is credited with the first proof that the set of primes is infinite: there is no largest prime as much as there is no largest natural number. The typical argument, a reasoning by absurdity, runs as follows. Let us suppose that the set of primes is finite, say  $\{p_1, p_2, p_3, \dots, p_n\}$ . Construct the number  $q = p_1 p_2 p_3 \dots p_n + 1$ . If  $q$  is a prime, then we have a new prime as  $q > p_i$  for all  $1 \leq i \leq n$ . If  $q$  is not a prime, then (by the prime factoring theorem) it must be divisible by a prime  $r < q$ . But  $r$  cannot be any  $p_i$  in our original exhaustive list of primes because dividing  $q$  by  $p_i$  produces the remainder 1. As a consequence,  $r$  is a new prime as well. In both cases we have found a contradiction, a prime that is not in the original list.

Wittgenstein criticized Euclid's proof on its "totality" nature because [4, p. 64]: "In mathematics we must always be dealing with systems, and not with totalities." (Wittgenstein advocated a form of anti-platonism by rejecting the interpretation of mathematical propositions in terms of propositions which are capable of being true or false in correspondence to reality, cf. [4].)

Instead of a proof of the existence of an infinity of primes based on deduction from certain formal or informal assumptions, Wittgenstein called for the construction of "a formal expression that proves the infinity of primes by its syntactical features."

Such a proof can be easily obtained by rephrasing Euclid's argument in terms of a formal or informal computer program  $\Pi_{\text{primes}}$  that generates the sequence of primes in increasing order. The infinity of primes is equivalent with the property of  $\Pi_{\text{primes}}$  to continue indefinitely and never stop. Although we do not subscribe to Wittgenstein's philosophy of mathematics, we acknowledge the importance of his preference for process versus function.

## 2.2 Goldbach's Conjecture and Other Problems in Number Theory

Goldbach's conjecture, which is part of Hilbert's eighth problem [5], states that all positive even integers greater than two can be expressed as the sum of two primes. The conjecture was tested up to  $10^{18}$ ; see [6].

Applying the same idea as for the infinity of primes, a computer program  $\Pi_{\text{Goldbach}}$  can be written that enumerates all positive even integers greater than two and for each of them checks the required property. The program  $\Pi_{\text{Goldbach}}$  stops if and only if it finds a counter-example for Goldbach's conjecture. In other words,  $\Pi_{\text{Goldbach}}$  never stops if and only if Goldbach's conjecture is true.

The same approach works for many problems in number theory, in particular for Fermat's last theorem. The program  $\Pi_{\text{Fermat}}$  systematically generates all 4-tuples of positive integers greater than 3,  $(n, x, y, z)$ , and stops when it finds the first 4-tuple for which  $x^n + y^n = z^n$ .

Can the same method be applied to the conjecture that there are infinitely many Mersenne primes? (Mersenne primes are numbers of the form  $2^n - 1$ . Currently only 46 Mersenne primes are known and the largest is  $2^{43\,112\,609} - 1$ . The conjecture is believed to be true because the harmonic series diverges.)

Or to the twin primes conjecture that there are infinitely many primes  $p$  such that  $p + 2$  is also prime? (This is believed to be true because of the probabilistic distribution of primes.)

It is clear that Goldbach's conjecture and Fermat's last theorem are statements of the form  $(\forall n) P(n)$ , where  $P$  is a computable predicate. The last two conjectures have a more complicated structure and can be written in the form  $(\forall N) (\exists n > N) P'(n)$ , where  $P'$  is a computable predicate. A program generating more and more natural numbers satisfying the twin primes conjecture may not stop either because there are infinitely many pairs of primes  $p, p + 2$  or because there are only finitely many primes  $p$  such that  $p + 2$  is also prime!

So, the last two conjectures cannot be directly represented by the halting property of an associated program. It is an open question whether the last conjectures can be represented in the form  $(\forall n) P(n)$ , where  $P$  is a computable predicate. Still, can they be described in

terms of the halting status of some program? A positive answer is given in Section 6.

### 2.3 The Pigeonhole Principle

The statement if  $n > m$  pigeons are put into  $m$  pigeonholes, there is a hole with more than one pigeon is called the pigeonhole principle or the Dirichlet principle.

Here is a more formal statement: every function from a set of  $n$  elements into a set with  $m < n$  elements is not injective. A program  $\Pi_{\text{pigeonholeprinciple}}$  which for all  $n$  generates all functions from  $\{1, 2, \dots, n\}$  into  $\{1, 2, \dots, m\}$ , for all  $m < n$ , and for each of them checks whether the function is injective, will either find an injective function and stop, or will continue forever. The validity of the pigeonhole principle is equivalent to the fact that the program  $\Pi_{\text{pigeonholeprinciple}}$  never stops.

### 2.4 Hilbert's Tenth Problem

Solving algebraic equations using integer (or rational) constants in the domain of positive integers is an old mathematical activity. Some of these equations do not have a solution at all, others have a finite number of solutions, and some have infinitely many solutions. The equation  $2x - 2y = 1$  has no integer solution, the equation  $5x = 10$  has a unique integer solution, and the equation  $7x - 17y = 1$  has infinitely many integer solutions.

A Diophantine equation (named after Diophantus of Alexandria, circa 200 AD) is an equation of the form  $P = 0$  where  $P$  is a polynomial with integer coefficients. Fermat's equations  $x^n + y^n = z^n$  for  $n = 1, 2, \dots$  are all Diophantine. To solve a given Diophantine equation  $P = 0$ , we have to determine whether the equation has solutions in the domain of positive integers, and, if it has, to find all of them.

Here is the formulation of Hilbert's tenth problem (for the original statement in German, see [7]):

10. Determining the solvability of a Diophantine equation. Given a Diophantine equation with any number of unknowns and with rational integer coefficients: devise a process, which could determine by a finite number of operations whether the equation is solvable in rational integers.

Consider the parametric Diophantine equation

$$(a_1, a_2, \dots, a_n, x_1, x_2, \dots, x_{m+1}) = 0, \quad (1)$$

where  $a_1, a_2, \dots, a_n$  are parameters and  $x_1, x_2, \dots, x_{m+1}$  are unknowns. Fixing values for parameters results in a particular Diophantine equation. For example, in the parametric Diophantine equation  $(a_1 - a_2)^2 - x_1 - 1 = 0$ ,  $a_1$  and  $a_2$  are the parameters and  $x_1$  is the

only unknown. If we put  $a_1 = 1$  and  $a_2 = 0$  we get the Diophantine equation  $x_1 = 0$ ; if we take  $a_1 = a_2 = 0$  we get the Diophantine equation  $x_1 + 1 = 0$ .

Given equation (1), we can construct a program  $\Pi_P$  which, beginning with the input  $a_1, a_2, \dots, a_n$ , will eventually halt if and only if equation (1) has a solution in the unknowns  $x_1, x_2, \dots, x_{m+1}$ . The program  $\Pi_P$  systematically generates all vectors with  $m + 1$  integers  $(i_1, i_2, \dots, i_{m+1})$ , checks for each of them whether  $P(a_1, a_2, \dots, a_n, i_1, i_2, \dots, i_{m+1}) = 0$ , and stops when the first solution is found.

Can we make the program  $\Pi_P$  “independent” of  $P$  in the sense that  $P$  appears as an input of the program? The answer is affirmative: we can construct a program  $\Pi_{H10P}$  that when given an arbitrary Diophantine equation (without parameters, i.e.,  $n = 0$ )  $P = 0$ , will eventually stop if and only if the equation  $P = 0$  has a solution. Can we decide in a finite amount of time whether the program  $\Pi_{H10P}$  eventually halts? The answer is negative, as is well known [8]. The core argument is based on the fact that every computably enumerable set of natural numbers that can be represented in the form  $\{n : P(n, x_1, x_2, \dots, x_m) = 0 \text{ has a solution in the non-negative integers unknown } x_1, x_2, \dots, x_m\}$ , cf. [9].

It is interesting to note that the counterpart of Hilbert’s tenth problem for real unknowns, that is, given an equation of the form  $P(x_1, x_2, \dots, x_m) = 0$  where  $P$  is a polynomial with integer coefficients (same as in the classical case) but  $x_1, x_2, \dots, x_m$  are real unknowns, is decidable. There is a program that decides in a finite amount of time whether the equation has a solution in the domain of reals. Indeed, the decision problem is solved by the Sturm method [10] for  $m = 1$  and Tarski’s method [11] works for any number of unknowns. This shows that extrapolating computational facts from positive integers to reals is not always possible. Of course, no program can in general compute exactly some solutions, even if their number is known, because solutions can be irrational. However, solutions can be effectively approximated up to any precision. This leads us to the following problem for standard Diophantine equations.

Fix a Diophantine equation

$$D(n, x_1, x_2, \dots, x_m) = 0, \quad (2)$$

and then consider the following two questions.

- For a fixed  $n = n_0$ , does the equation  $D(n_0, x_1, x_2, \dots, x_m) = 0$  have a solution?
- For a fixed  $n = n_0$ , does the equation  $D(n_0, x_1, x_2, \dots, x_m) = 0$  have an infinity of solutions?

Both questions are undecidable for *some* instances of equation (2). For *each* equation (2) the information contained in the sequence of  $k$  answers to the yes/no question “does the equation  $D(n_0, x_1, x_2, \dots, x_m) = 0$  have a solution for  $n = 1, 2, \dots, k$ ?” contains only  $\log k$  bits of information (knowing how many equations have solutions is enough to determine exactly which equations have solutions). This information can be substantially compressed. However, for some instances of equation (2) the information contained in the sequence of  $k$  answers to the yes/no question “does the equation  $D(n_0, x_1, x_2, \dots, x_m) = 0$  have infinitely many solutions for  $n = 1, 2, \dots, k$ ?” contains about  $k$  bits of information, that is, the information cannot be algorithmically compressed. For more, see [12].

## 2.5 The Four Color Theorem

The four color theorem, first conjectured in 1853 by Francis Guthrie, states that every plane separated into regions may be colored using no more than four colors in such a way that no two adjacent regions receive the same color. Two regions are called *adjacent* if they share a border segment, not just a point. Regions must be contiguous, that is, the plan has no exclaves.

In graph-theoretical terms, the four color theorem states that the vertices of every planar graph can be colored with at most four colors so that no two adjacent vertices receive the same color. Shortly, every planar graph is four-colorable.

The four color theorem was proved in 1977 [13, 14] (see also [15]) using a computer-assisted proof that consists of constructing a finite set of “configurations”, and then proving that each of them is “reducible”—which implies that no configuration with this property can appear in a minimal counter-example to the theorem. Checking the correctness of the original proof is a very difficult task. It implies, among other things, checking the descriptions of 1476 graphs, checking the correctness of the programs, proving the correctness of the compiler used to compile the programs, and checking the degree of reliability of the hardware used to run the programs. This computer-assisted proof generated lots of mathematical and philosophical discussions around the notion of acceptable mathematical proof; see, for example, [16-18]. Various partial independent verifications have been obtained, though it appears that there is no verification in its entirety. The formal confirmation announced in [19] uses the equational logic program Coq (see [20] for a recent presentation of the formal proof). The following quote from the concluding discussion in [19] is relevant for the current status of the proof (emphasis added):

However, an argument can be made that our “proof” is not a proof in the traditional sense, because it contains steps that can never be verified by humans. In particular, we have not proved the correctness of the compiler we compiled our programs on, nor have we proved the infallibility of the hardware we ran our programs on. These have to be taken on faith, and are conceivably a source of error.... Apart from this hypothetical possibility of a computer consistently giving an incorrect answer, the rest of our proof can be verified in the same way as traditional mathematical proofs. We concede, however, that *verifying a computer program is much more difficult than checking a mathematical proof of the same length.*

A program  $\Pi_{\text{fourcolortheorem}}$  that systematically generates all planar graphs and checks if each one is colorable with four colors and stops when the first counter-example is found will never halt if and only if the theorem is true. However, this program will be quite long because testing the planarity of a graph is difficult. A better solution is to use the Diophantine representation of the four color theorem proposed in [9]:

$$F(n, t, a, \dots) = 0. \quad (3)$$

Equation (3) has no solution if and only if every planar graph can be colored with at most four colors so that no two adjacent vertices receive the same color. Based on equation (3), we can write the program  $\Pi_F$  as in Section 2.4, which can be taken as  $\Pi_{\text{fourcolortheorem}}$ .

Actually, it is better to use a pre-Diophantine representation given by the following conditions. Without restricting the generality, we consider the maps  $T_n$  consisting of the points  $(x, y)$  such that  $J(x, y) \leq Q = (n^2 + 3n)/2$ , where  $J$  is Cantor’s bijection  $J(x, y) = ((x+y)^2 + 3x+y)/2$ . Given a four-coloring of  $T_n$ ,  $t_0, t_1, \dots, t_Q$  there exist (and can be effectively computed)  $s, t$  such that for all  $0 \leq i \leq Q$  (the integer remainder function is denoted by  $\text{rem}$ ):

$$t_i = \text{rem}(t, 1 + s(i+1)).$$

In other words, the sequence  $t_0, t_1, \dots, t_Q$  can be coded by  $s$  and  $t$ .

Every sequence  $u_0, u_1, \dots, u_Q$  with  $u_i < 4$  can be represented by some  $u \leq R = (1 + 4(Q+2)!)^{Q+1}$  such that

$$u_i = \text{rem}(u, 1 + 4(Q+2)!(i+1)).$$

Finally, there is a map (say,  $T_n$ ) that cannot be colored in four colors if and only if the following condition is satisfied:

$$(\exists n, t, s) (\forall u \leq R) (\exists x, y) (x + y \leq n) [A \vee B],$$

where

$$A = u_J(x,y) \geq 4,$$

$$B = \left[ \left( t_J(x,y) = t_J(x+1,y) \wedge u_J(x,y) \neq u_J(x+1,y) \right) \vee \right. \\ \left. \left( t_J(x,y) \neq t_J(x+1,y) \wedge u_J(x,y) = u_J(x+1,y) \right) \vee \right. \\ \left. \left( t_J(x,y) = t_J(x,y+1) \wedge u_J(x,y) \neq u_J(x,y+1) \right) \vee \right. \\ \left. \left( t_J(x,y) \neq t_J(x,y+1) \wedge u_J(x,y) = u_J(x,y+1) \right) \right].$$

A simple inspection shows that the given condition is computable, so the four color theorem is of the form  $(\forall n) P(n)$ , where  $P$  is a computable predicate.

## 2.6 The Riemann Hypothesis

The Riemann hypothesis is probably the most famous and important conjecture in mathematics. It appears in Hilbert's eighth problem [5]: the nontrivial complex zeros of Riemann's zeta function, which is defined for  $\text{Re}(s) > 1$  by

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s},$$

lie exactly on the line  $\text{Re}(s) = 1/2$ .

According to Matiyasevich [8, pp. 119-121], the negation of the Riemann hypothesis is equivalent to the existence of positive integers  $k, l, m, n$  satisfying the following six conditions (here  $x | z$  means “ $x$  divides  $z$ ”):

1.  $n \geq 600$ ,
2.  $\forall y < n [(y+1) | m]$ ,
3.  $m > 0 \ \& \ \forall y < m [y = m \vee \exists x < n [\neg [(x+1) | y]]]$ ,
4.  $\text{explog}(m-1, l)$ ,
5.  $\text{explog}(n-1, k)$ ,
6.  $(l-n)^2 > 4n^2 k^4$ ,

and  $\text{explog}(a, b)$  denotes the predicate

$$\exists x \left[ x > b+1 \ \& \ \left( 1 + \frac{1}{x} \right)^{xb} \leq a+1 < 4 \left( 1 + \frac{1}{x} \right)^{xb} \right].$$

An inspection of these conditions shows that the Riemann hypothesis is of the form  $\forall n, R(n)$ , where  $R$  is a computable predicate. Hence,



one can write a program  $\Pi_{\text{Riemann}}$  such that the Riemann hypothesis is false if and only if  $\Pi_{\text{Riemann}}$  halts.

## 2.7 The Collatz and Palindrome Conjectures

When he was a student, L. Collatz posed the following problem: given any integer seed  $a_1$ , there exists a natural  $N$  such that  $a_N = 1$ , where

$$a_{n+1} = \begin{cases} \frac{a_n}{2}, & \text{if } a_n \text{ is even,} \\ 3a_n + 1, & \text{otherwise.} \end{cases}$$

This is known as Collatz's conjecture, the Syracuse conjecture, the  $3x + 1$  problem, Kakutani's problem, Hasse algorithm, or Ulam's problem. There is a huge amount of literature on this problem and various natural generalizations: see [21-23]. Erdős has said (cf. [21]) that "Mathematics may not be ready for such problems."

Does there exist a program  $\Pi_{\text{Collatz}}$  such that Collatz's conjecture is false if and only if  $\Pi_{\text{Collatz}}$  halts?

First, we note that a brute-force tester, that is, a program that will enumerate all seeds and try to find an iteration equal to 1 for each of them, may never stop in two different cases: (a) because the conjecture is indeed true, or (b) because for some specific seed  $a_1$  there is no  $N$  such that  $a_N = 1$ . It is not clear how to differentiate these cases; even worse, it is not clear how to refute (b) using a brute-force tester.

A simple nonconstructive argument answering our question in the affirmative appears in [3]. Indeed, observe first that the set

$$\text{Collatz} = \{a_1 : a_N = 1, \text{ for some } N \geq 1\}$$

is computably enumerable. Collatz's conjecture requires proving that *Collatz* does indeed contain all positive integers.

If *Collatz* is not computable, then the conjecture is false, and any program that eventually halts can be taken as  $\Pi_{\text{Collatz}}$  as (a) is ruled out. If *Collatz* is computable, then we can write a program  $\Pi_{\text{Collatz}}$  to find an integer not in *Collatz*: the conjecture is true if and only if  $\Pi_{\text{Collatz}}$  never stops.

Now we present the palindrome conjecture. The reverse (mirror) of a number is the number formed with the same decimal digits but written in the opposite order. For example, the mirror of 12 is 21, the mirror of 131072 is 270131, and so on. Start with the decimal representation of a natural  $a$ , reverse the digits, and add the constructed number to  $a$ . Iterate this process until the result is a palindrome. Following [24], the palindrome conjecture states that for every natural  $a$ , a palindrome number will be obtained after finitely many iterations of the given procedure.

The same nonconstructive argument used for Collatz's conjecture applies to the palindrome conjecture: there exists a program  $\Pi_{\text{palindrome}}$  such that the palindrome conjecture is true if and only if  $\Pi_{\text{palindrome}}$  never stops.

Collatz and palindrome conjectures have the following general form. Let  $a \in \mathbf{N}$  and let  $T$  be a computable function from naturals to naturals. The conjecture associated to  $(a, T)$  is: for each  $x \in \mathbf{N}$ ,  $T^i(x) = a$ , for some  $i > 0$ .

Considering the set

$$B(a, T) = \{x \in \mathbf{N} : T^i(x) = a, \text{ for some } i > 0\}, \quad (4)$$

the conjecture associated to  $(a, T)$  becomes equivalent with the equality  $B(a, T) = \mathbf{N}$ . The argument used for Collatz's conjecture applies to this general case too, so one can prove in a nonconstructive way the existence of a program  $\Pi_{(a,T)}$  such that the conjecture associated to  $(a, T)$  is true if and only if  $\Pi_{(a,T)}$  never stops.

### 3. The Halting Problem

---

In Section 2 the halting property of various programs repeatedly appeared. It is time to ask the question: can the halting problem be solved by a program? As all of our programs have a very specific form—they have no input and each of them either stops (in which case the output is a natural number) or never stops—we show with a simple argument that the halting problem, that is, the problem of whether or not such a program eventually stops, is unsolvable by any program. This means that there is no program **H** with the following three properties.

1. **H** accepts as input any program of the given type.
2. **H** eventually halts.
3. **H** produces the output 1 if the input program eventually stops, or 0 in case the input program never stops.

Here is an information-theoretic analysis of the existence of the hypothetical program **H**. Assume that there exists a halting program **H** with the three properties. Using **H** we construct the following (legitimate) program **P**.

1. Read a natural  $N$ .
2. Generate all programs up to  $N$  bits in size.
3. Use **H** to check each generated program for whether or not it halts.

4. Simulate the running of the remaining programs.
5. Output 1 plus the biggest value output by these programs.

The program  $\mathbf{P}$  halts for every natural  $N$ . Indeed, the number of programs less than  $N$  bits in size is finite, so by the assumption that  $\mathbf{H}$  can decide the halting status of every program in a finite amount of time, we can filter out all nonhalting programs. The remaining programs (certainly, finitely many) can be run and after a finite (maybe very long) computation they will all halt and each will produce a natural number as output. Did we obtain a contradiction?

To answer the question we have to ask the right auxiliary question: how long is  $\mathbf{P}$ ? The answer is that  $\mathbf{P}$  is about  $\log_2 N$  bits. Indeed, we need about  $\log_2 N$  bits to code  $N$  in binary, and the rest of the program  $\mathbf{P}$  is a constant, say  $c$ . Hence, the length of  $\mathbf{P}$  is  $\log_2 N + c$  bits.

Now observe that there is a big difference between the size of  $\mathbf{P}$  and the size of the output produced by  $\mathbf{P}$ . Indeed, for large enough  $N$ ,  $\mathbf{P}$  belongs to the set of programs having less than  $N$  bits because  $\log_2 N + O(1) < N$ . Hence, in this case,  $\mathbf{P}$  generates itself at some stage of the computation. As  $\mathbf{P}$  always halts, the program  $\mathbf{H}$  decides that  $\mathbf{P}$  stops, so  $\mathbf{P}$  is run as a part of the simulation (inside the computation of  $\mathbf{P}$ ) and produces a natural number as a result. But the program  $\mathbf{P}$  itself will output a natural number that is different from every output produced by a simulated computation, in particular from the output produced by  $\mathbf{P}$  itself, which is a contradiction.

This proof (see [12] for more details) shows that in general there is no method, no uniform procedure, to test whether an arbitrary program eventually stops or not. Of course, this does not imply that for some infinite class of programs we cannot find a program to decide their halting status. Actually, there are infinite sets of programs for which the halting problem is decidable, for example, the class of primitive recursive programs (which are all total).

What is the situation with the programs associated to the problems discussed before? Can we hope to decide for each of them whether it halts or not? For some programs, such as  $\Pi_{\text{Fermat}}$ , we know the answer: the program never stops as certified by A. Wiles' proof of Fermat's last theorem. For the program  $\Pi_{\text{Riemann}}$  the answer is not known. We currently cannot even explicitly write the program  $\Pi_{\text{Collatz}}$ . (We conjecture that the statement " $\Pi_{\text{Collatz}}$  never stops" is independent of the Zermelo-Fraenkel set theory with the axiom of choice, or ZFC.) For some programs  $\Pi$  the statement " $\Pi$  halts" is independent of ZFC. Such a statement has to be true. A proof of independence is an alternative proof, admittedly not usual, of the truth of the statement.

#### 4. A Computational Method for Evaluating the Complexity of Mathematical Problems

We now return to Fermat's last theorem and to the fact that it is equivalent to the statement " $\Pi_{\text{Fermat}}$  never halts". We do not propose to prove Fermat's last theorem by showing that  $\Pi_{\text{Fermat}}$  never halts, but to use the program  $\Pi_{\text{Fermat}}$  as a way to measure the complexity of Fermat's last theorem.

We do this by counting the number of bits necessary to specify  $\Pi_{\text{Fermat}}$  in some fixed "universal formalism" (e.g., a universal self-delimiting Turing machine [12]). Of course, there are many programs equivalent to  $\Pi_{\text{Fermat}}$ , so a natural way to evaluate their complexity is to consider the smallest such program.

The choice of the universal formalism used to code programs is irrelevant up to an additive constant, so if a problem is significantly more complex in some fixed formalism than in another one, then it will continue to be more complex in any other formalism. However, the proposed measure is uncomputable [12], so we have to work with an upper bound on the size of a program "describing" the conjecture, problem, or theorem.

In practice, to evaluate the complexity of a problem  $P$  we need to effectively obtain the program  $\Pi_P$  and compute its size in bits. This gives an upper bound on the complexity of  $\Pi_P$ , and hence on the difficulty of  $P$ . But, as we have seen with the Collatz and palindrome conjectures, even this type of approximation may not be achievable in all cases. We cannot evaluate a bound on the difficulty of a problem  $P$  if we do not know at least one "explicit" program  $\Pi_P$ .

#### 5. Finitely Refutable Problems

It is time to ask the question: What is the class of problems whose complexity can be evaluated with the method proposed in Section 4? We will not answer this question, which is open, but do give an example of a large class of problems to which the method applies. With Pythagoras' dictum "all is number" as a guiding principle we will look at finite numerical tests.

Let  $\mathbf{N}$  denote the set of positive integers and for every  $k \in \mathbf{N}$  consider a predicate  $P$  on  $\mathbf{N}$ .

Consider the formula

$$f = Q_1 n_1 Q_2 n_2 \dots Q_k n_k P(n_1, n_2, \dots, n_k)$$

where  $Q_1, Q_2, \dots, Q_k \in \{\forall, \exists\}$  are quantifier symbols. In analogy with the arithmetic classes, we say that  $f$  is in the class  $\hat{\Pi}_s$  or  $\hat{\Sigma}_s$  if the quantifier prefix of  $f$  starts with  $\forall$  or  $\exists$ , respectively, and contains

$s - 1$  alternations of quantifier symbols. When  $P$  is computable, then  $f$  is in  $\Pi_s$  or  $\Sigma_s$ , respectively. It is sufficient to consider only such formulas  $f$  in which no two consecutive quantifier symbols are the same. In the remainder of this section we make this assumption without special mention. With  $f$  as given, we have  $s = k$ .

As usual, with  $P$  as given, we write  $P(n_1, \dots, n_k)$  instead of  $P(n_1, \dots, n_k) = 1$  when  $n_1, \dots, n_k$  are elements of  $\mathbf{N}$ . Thus,  $\neg P(n_1, \dots, n_k)$  if and only if  $P(n_1, \dots, n_k) = 0$ . Moreover, since we consider variable symbols only in the domain  $\mathbf{N}$ , if  $f$  is any formula in first-order logic, we write  $f$  is true instead of  $f$  is true in  $\mathbf{N}$ .

Let  $\Gamma_s$  be one of the classes  $\hat{\Pi}_s, \hat{\Sigma}_s, \Pi_s$ , or  $\Sigma_s$ . We refer to the task of proving or refuting a first-order logic formula as a *problem* and, in particular, to problems expressed by formulas in  $\Gamma_s$  as  $\Gamma_s$ -*problems*.

We say that a problem is being *solved* if the corresponding formula is proved or disproved to be true, that is, if the truth value of the formula is determined. A problem is said to be *finitely solvable* if it can be solved by examining finitely many cases.

For example, consider the predicate

$$P(n) = \begin{cases} 1, & \text{if } n \text{ is even or } n = 1 \text{ or } n \text{ is a prime,} \\ 0, & \text{otherwise,} \end{cases}$$

that is,  $P(n) = 0$  if and only if  $n$  is an odd number greater than 1 which is not a prime. Then the conjecture expressed by the formula  $(\forall n)P(n)$  is finitely solvable; indeed, it is sufficient to check all  $n$  up to 10 to refute this conjecture.

Goldbach's conjecture is a  $\Pi_1$ -problem. To express it let  $P_{\text{Goldbach}} : \mathbf{N} \rightarrow \{0, 1\}$  be such that

$$P_{\text{Goldbach}}(n) = \begin{cases} 1, & \text{if } n \text{ is odd or } (n \text{ is even and } n \text{ is the sum of two primes),} \\ 0, & \text{otherwise.} \end{cases}$$

Thus,  $f_{\text{Goldbach}} = (\forall n)P_{\text{Goldbach}}(n)$  is true if and only if Goldbach's conjecture is true.

Similarly, the Riemann hypothesis is a  $\Pi_1$ -problem. By a result given in [9], the Riemann hypothesis can be expressed in terms of the function  $\delta_{\text{Riemann}} : \mathbf{N} \rightarrow \mathbf{R}$  defined by

$$\delta_{\text{Riemann}}(k) = \prod_{n < k} \prod_{j \leq n} \eta_{\text{Riemann}}(j),$$

where

$$\eta_{\text{Riemann}}(j) = \begin{cases} p, & \text{if } j = p^r \text{ for some prime } p \text{ and some } r \in \mathbf{N}, \\ 1, & \text{otherwise.} \end{cases}$$

The Riemann hypothesis is equivalent with the assertion that for all  $n \in \mathbf{N}$

$$\left( \sum_{k \leq \delta_{\text{Riemann}}(n)} \frac{1}{k} - \frac{n^2}{2} \right)^2 < 36 n^3.$$

If we set

$$P_{\text{Riemann}}(n) = \begin{cases} 1, & \text{if } \left( \sum_{k \leq \delta_{\text{Riemann}}(n)} \frac{1}{k} - \frac{n^2}{2} \right)^2 < 36 n^3, \\ 0, & \text{otherwise.} \end{cases}$$

Then,  $f_{\text{Riemann}} = (\forall n) P_{\text{Riemann}}(n)$  is true if and only if the Riemann hypothesis is true. Clearly,  $P_{\text{Riemann}}$  is decidable, therefore, the Riemann hypothesis is a  $\Pi_1$ -problem.

What is the ‘‘commonality’’ of all problems in classes  $\hat{\Pi}_s$  and  $\hat{\Sigma}_s$ ?

For  $s \in \mathbf{N}$ , let  $\hat{\Gamma}_s$  denote any of  $\hat{\Pi}_s$  and  $\hat{\Sigma}_s$ , and let  $\Gamma_s$  denote any of  $\Pi_s$  and  $\Sigma_s$ . Let

$$f = Q_1 n_1 Q_2 n_2 \dots Q_s n_s P(n_1, n_2, \dots, n_s)$$

with  $s \in \mathbf{N}$ , where  $Q_1, Q_2, \dots, Q_s$  are alternating quantifier symbols.

Following [25], we define a test set for  $f$  to be a set  $T \subseteq \mathbf{N}^s$  such that  $f$  is true in  $\mathbf{N}^s$  if and only if it is true in  $T$ . The problem  $f$  is finitely solvable if there is a finite test set for  $f$ . In [25] the following result was proved:

*Every  $f \in \hat{\Gamma}_s$  is finitely solvable.*

In other words, a solution to each mass problem (i.e., a problem having an infinite number of instances or cases) in the given classes can be obtained by inspecting only finitely many instances of the problem. As we might expect, this fact cannot be used to obtain a uniform algorithmic way of solving these types of problems because the finite test set cannot be computed even for all problems in the class  $\Pi_1$ : There is no constructive proof showing that every  $f \in \Pi_1$  has a finite test set.

## 6. The Power and the Limits of the Method

Our analysis gives a new method of comparing the difficulties of two or more finitely refutable problems. The main obstacle is the noncomputability of the measure [12]. However, by working with upper bounds, we can obtain a practical method for evaluating the complexity that allows a relative ranking of problems.

In weighting the importance of computing the *exact value* of the complexity measure, recall Knuth [26]: “premature optimization is the root of all evil” and Rabin [27] “we should give up the attempt to derive results and answers with complete certainty.”

The method can be applied to every  $\Pi_1$ -problem. All problems discussed in Section 2 can be analyzed with this method [3]. Trying to reduce the length of a program is in general possible; of course, proving minimality is, in general, impossible [12].

The method proposed is certainly not universal. Let us discuss here the class of  $\Pi_1$ -problems. Not every mathematical statement is a  $\Pi_1$ -problem. For instance, the twin primes conjecture discussed in Section 2.2 is not a  $\Pi_1$ -problem. Writing

$$P_{\text{TP}}(n, m) = \begin{cases} 1, & m > n \text{ and } m \text{ and } m + 2 \text{ are primes,} \\ 0, & \text{otherwise,} \end{cases}$$

this conjecture can be stated as

$$f_{\text{TP}} = \forall n \exists m P_{\text{TP}}(n, m).$$

The formula  $f_{\text{TP}}$  is in the class  $\Pi_2$ . Bennett conjectured in [28] that most mathematical conjectures can be settled indirectly by proving stronger conjectures. For the twin primes conjecture a stronger  $\Pi_1$ -problem is obtained as follows. Consider the predicate

$$P'_T(n) = \begin{cases} 1, & \text{if there is } m \text{ with } 10^{n-1} \leq m \leq 10^n, m \text{ and } m + 2 \text{ primes,} \\ 0, & \text{otherwise.} \end{cases}$$

Let  $f'_T = (\forall n) P'_T(n)$ . Thus,  $f'_T$  gives rise to a  $\Pi_1$ -problem and, if  $f'_T$  is true, then  $f_T$  is also true (but the converse is not necessarily true).

However, there exists a program  $\Pi_{\text{TP}}$  such that the twin primes conjecture is true if and only if  $\Pi_{\text{TP}}$  never halts. As in Collatz's case, the argument is nonconstructive and based on the fact that the set

$$\text{TP} = \{n : \forall n \exists m > n \text{ such that } m \text{ and } m + 2 \text{ are primes}\}$$

is computably enumerable.

The method depends on the chosen universal Turing machine, that is, our working framework. Changing the universal machine will result in changes of the complexity value, but not in relative compari-

son between problems. The choice of the machine is irrelevant up to an additive constant, so if a problem is significantly more complex than another one with respect to a fixed universal machine, then it will continue to be more complex for any other machine. The method was used by relativizing to the halting problem. The same method can be used by relativizing to other unsolvable problems, for example, the totality problem. Going from the halting problem to the totality problem will increase the power of expression. For example, to the conjecture associated to  $(a, T)$  (see equation (4)), we can associate the program  $\Gamma_{a,T}(x)$  defined by

$$\Gamma_{a,T}(x) = \min_i [T^i(x) = a].$$

The conjecture associated to  $(a, T)$  is true if and only if  $\Gamma_{a,T}(x)$  is total. Writing the program for  $\Gamma_{a,T}(x)$  is simple, but in the special cases of the Collatz, palindrome, and twin primes conjectures writing the corresponding  $\Pi_{a,T}$  program is problematic (we are only able to prove its existence).

In [29] Kim discusses the possibility that the Poincaré conjecture (a recently solved problem in topology, see [30-32]) is equivalent to the unsolvability of a Diophantine equation (see also [33]). If this is true, then our method would also offer, at least in principle, an indication of the difficulty of the Poincaré conjecture.

## 7. Conclusions

We have presented a computational method for evaluating the complexity of mathematical problems. The method, inspired by [1], is based on the possibility of completely describing complex mathematical problems, such as the Riemann hypothesis, in terms of very simple programs.

If a mathematical problem, irrespective of its nature, can be equivalently expressed in terms of the property that a certain associated program eventually halts, then the proposed method applies. For example, the method applies to every  $\Pi_1$ -problem. Specific instances of such problems are, for example, Fermat's last theorem, the Goldbach conjecture, the four color problem, the Riemann hypothesis, Hilbert's tenth problem, the Collatz problem, the palindrome conjecture, and the twin primes conjecture. As an illustration, according to this complexity measure, the Riemann hypothesis is about twice as difficult as the Goldbach conjecture [3]. Although the method applies to both the Collatz and twin primes conjectures, it is an open question whether one can effectively evaluate the complexity of these problems.

Our method, which is a refinement below the first Turing degree, provides a total order in the class of finitely refutable problems. The



difficulty of the problem is additive (modulo the constants involved due to the choice of the universal Turing machine). The same method can be used by relativizing to other unsolvable problems different from the halting problem (e.g., the totality problem).

The scalability of the measure, both in terms of ordering, the role of the additive constants involved, and its relativization to various unsolvable problems are open questions.

In Part 2 of this study we will present a formalism for uniformly evaluating the complexity of the problems discussed in this paper and a ranking of those problems will be presented.

## Acknowledgment

---

We thank Professor John Casti for discussions and comments regarding the topic of this paper which improved our presentation. We also thank the anonymous referee for very helpful comments and suggestions.

This work was supported in part by the Andrea von Braun Foundation, Munich, under the grant for “Artistic Forms and Complexity”.

## References

---

- [1] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.
- [2] “2010 Mathematics Subject Classification (MSC2010).” (Sep 17, 2009) [www.ams.org/msc](http://www.ams.org/msc).
- [3] C. S. Calude, E. Calude, and M. J. Dinneen, “A New Measure of the Difficulty of Problems,” *Journal for Multiple-Valued Logic and Soft Computing*, 12(3-4), 2006 pp. 285-307.
- [4] T. Lampert, “Wittgenstein on the Infinity of Primes,” *History and Philosophy of Logic*, 29(1), 2008 pp. 63-81. doi.10.1080/01445340701507569.
- [5] D. Hilbert, “Mathematical Problems,” *Bulletin of the American Mathematical Society*, 8(10), 1902 pp. 437-479.
- [6] T. Oliveira e Silva. “Goldbach Conjecture Verification.” (Jul 28, 2009) [www.ieeta.pt/~tos/goldbach.html](http://www.ieeta.pt/~tos/goldbach.html).
- [7] Y. V. Matiyasevich. “Hilbert’s Tenth Problem.” (Nov 30, 2008) [logic.pdmi.ras.ru/Hilbert10/stat/stat-eng.htm](http://logic.pdmi.ras.ru/Hilbert10/stat/stat-eng.htm).
- [8] Y. V. Matiyasevich, *Hilbert’s Tenth Problem*, Cambridge, MA: The MIT Press, 1993.
- [9] M. Davis, Y. V. Matiyasevich, and J. Robinson, “Hilbert’s Tenth Problem. Diophantine Equations: Positive Aspects of a Negative Solution,” *Mathematical Developments Arising from Hilbert Problems* (F. E. Browder, ed.), Providence, RI: American Mathematical Society, 1976 pp. 323-378.

- [10] E. W. Weisstein. “Sturm Function” from Wolfram *MathWorld*—A Wolfram Web Resource. [mathworld.wolfram.com/SturmFunction.html](http://mathworld.wolfram.com/SturmFunction.html).
- [11] A. Tarski, *A Decision Method for Elementary Algebra and Geometry*, 2nd ed., Berkeley: University of California Press, 1951.
- [12] C. S. Calude, *Information and Randomness: An Algorithmic Perspective*, 2nd ed. (revised and extended), Berlin: Springer-Verlag, 2002.
- [13] K. Appel, W. Haken, and J. Koch, “Every Planar Map Is Four-Colorable, I: Discharging,” *Illinois Journal of Mathematics*, **21**(3), 1977 pp. 429-490.
- [14] K. Appel and W. Haken, “Every Planar Map Is Four-Colorable, II: Reducibility,” *Illinois Journal of Mathematics*, **21**(3), 1977 pp. 491-567.
- [15] R. Wilson, *Four Colours Suffice: How the Map Problem Was Solved*, London: Penguin, 2002.
- [16] A. S. Calude, “The Journey of the Four Colour Theorem through Time,” *The New Zealand Mathematics Magazine*, **38**(3), 2001 pp. 27-35.
- [17] C. S. Calude, E. Calude, and S. Marcus, “Passages of Proof,” *Bulletin of the European Association for Theoretical Computer Science*, **84**, 2004 pp. 167-188.
- [18] C. S. Calude, E. Calude, and S. Marcus, “Proving and Programming,” *Randomness and Complexity: from Leibniz to Chaitin* (C. S. Calude, ed.), Singapore: World Scientific, 2007 pp. 310-321.
- [19] N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas. “The Four Color Theorem.” (Nov 30, 2008) [www.math.gatech.edu/~thomas/FC/fourcolor.html](http://www.math.gatech.edu/~thomas/FC/fourcolor.html).
- [20] G. Gonthier, “Formal Proof—The Four Color Theorem,” *Notices of the American Mathematical Society*, **55**(11), 2008 pp. 1382-1393.
- [21] J. C. Lagarias, “The  $3x + 1$  Problem and Its Generalizations,” *American Mathematical Monthly*, **92**, 1985 pp. 3-23.
- [22] R. K. Guy, “Problem E16,” *Unsolved Problems in Number Theory*, 3rd ed., (R. K. Guy, ed.), New York: Springer, 2004 pp. 330-336.
- [23] J.-P. Davaian, “ $3x + 1$ , Collatz, Syracuse Problem.” (Nov 30, 2008) [pagesperso-orange.fr/jean-paul.davaian/liens/liens\\_syracuse.html](http://pagesperso-orange.fr/jean-paul.davaian/liens/liens_syracuse.html).
- [24] J.-P. Delahaye, “Déconcertantes Conjectures,” *Pour la Science*, **367**, 2008 pp. 90-95.
- [25] C. S. Calude, H. Jürgensen, and S. Legg, “Solving Finitely Refutable Mathematical Problems,” *Finite versus Infinite. Contributions to an Eternal Dilemma* (C. S. Calude and G. Păun, eds.), London: Springer-Verlag, 2000 pp. 39-52.
- [26] D. E. Knuth, “Structured Programming with Go To Statements,” *ACM Computing Surveys (CSUR)*, **6**(4), 1974 pp. 261-301. [doi.acm.org/10.1145/356635.356640](https://doi.org/10.1145/356635.356640).
- [27] M. O. Rabin, “The Possibilities of Chance,” *Out of their Minds: The Lives and Discoveries of 15 Great Computer Scientists* (D. Sasha and C. Lazare, eds.), New York: Copernicus, 1995 pp. 68-89.
- [28] C. H. Bennett, “Chaitin’s Omega,” *Fractal Music, Hypercards, and More —: Mathematical Recreations from Scientific American Magazine* (M. Gardner, ed.), New York: W. H. Freeman, 1992 pp. 307-319.

- [29] M. Kim. “Why Everyone Should Know Number Theory.” (Nov 30, 2008) [www.ucl.ac.uk/~ucahmki/numbers.pdf](http://www.ucl.ac.uk/~ucahmki/numbers.pdf).
- [30] B. Kleiner and J. Lott. “Notes and Commentary on Perelman’s Ricci Flow Papers.” (Nov 30, 2008) [math.berkeley.edu/~lott/ricciflow/perelman.html](http://math.berkeley.edu/~lott/ricciflow/perelman.html).
- [31] G. Perelman. “The Entropy Formula for the Ricci Flow and Its Geometric Application.” (Nov 30, 2008) [www.arxiv.org/abs/math.DG/0211159](http://www.arxiv.org/abs/math.DG/0211159).
- [32] G. Perelman, “Ricci Flow with Surgery on Three-Manifolds.” (Nov 30, 2008) [www.arxiv.org/abs/math.DG/0303109](http://www.arxiv.org/abs/math.DG/0303109).
- [33] J. F. Manning, “Algorithmic Detection and Description of Hyperbolic Structures on Closed 3-Manifolds with Solvable Word Problem,” *Geometry & Topology*, **6**, 2002 pp. 1-26. doi .10.2140/gt.2002.6.1.