*J. of Mult.-Valued Logic & Soft Computing.*, Vol. 12, pp. 285–307
Reprints available directly from the publisher
Photocopying permitted by license only


# A New Measure of the Difficulty of Problems

CRISTIAN S. CALUDE, ELENA CALUDE AND MICHAEL J. DINNEEN

*Department of Computer Science, University of Auckland, New Zealand*
*E-mail: cristian@cs.auckland.ac.nz*
*IIMS, Massey University at Albany, New Zealand*
*E-mail: e.calude@massey.ac.nz*
*Department of Computer Science, University of Auckland, New Zealand*
*E-mail: mjd@cs.auckland.ac.nz*

*Received: February 16, 2006.*

Guessing the degree of difficulty of a problem before seeing its solution is notoriously hard not only for beginners, but also for the most experienced professionals. Can we develop a method to evaluate, in some objective way, the difficulty of an open problem? This note proposes such a measure which can be used for a fairly large class of finitely refutable conjectures which includes, for example, Riemann Hypothesis and the Goldbach's Conjecture. According to our measure, Riemann Hypothesis is more complex than Goldbach's Conjecture. We also show, in a non-constructive way, that the Collatz $3x + 1$ Problem is finitely refutable; consequently, our method cannot be applied, hence stronger versions of this problem are studied.

## 1 INTRODUCTION

Guessing the degree difficulty of a problem before seeing its solution is notoriously hard not only for beginners, but also for the most experienced professionals.

The best illustration is the famous list of 23[1] open problems proposed by Hilbert at the International Mathematical Congress in Paris on August 8, 1900 (see [1, 13, 16]), widely considered the most influential set of problems for the last century mathematics.

Hilbert's third problem, "give two tetrahedra that cannot be decomposed into congruent tetrahedra directly or by adjoining congruent tetrahedra", was

[1] Actually, 24, see [21].

solved by Dehn [8] immediately after the Congress, and the paper including its solution was published before the publication of the proceedings of the Congress. This may suggest that the problem was not "that difficult".

Hilbert's eitgth problem includes two problems neither which was proved nor disproved: the Riemann Hypothesis (one of the Clay Mathematical Institute Millennium Problems, [22]; see more about this problem in [12, 16–18]); its first solution will be awarded a prize worth 1 million dollars) and the Goldbach's Conjecture. While the Riemann Hypothesis is more difficult to explain, Goldbach's Conjecture, written in a June 7, 1742 letter to Euler, is easy to state: prove that "every positive even integer can be expressed as the sum of two primes," [10, 11].[2]

Finally, our last open question is Collatz $3x + 1$ Problem, about which Erdös commented that "mathematics is not yet ready for such problems", [14]. The problem asks to prove (or disprove) that for every seed $a > 0$, there is an iteration $N$ of the function $T(x) = x/2$, if $x$ is even, and $T(x) = 3x + 1$, if $x$ is odd, such that $T^N(a) = 1$.

Is there any mathematical way to evaluate the degree of difficulty of any of these problems? In what follows we will propose a uniform method to evaluate the difficulty of a finitely refutable problem. A conjecture is finitely refutable if verifying a finite number of instances suffices to disprove it (see [4]). Our refutable method uses a systematic enumeration (of the problem's search domain) that is guaranteed to find a counter-example if one exists. If the search stops, then it refutes the conjecture; if the search does not halt, then the conjecture is true.

Both Riemann's Hypothesis and Goldbach's Conjecture are finitely refutable, so our method allows the evaluation of their difficulty: according to our measure, the Riemann Hypothesis is more difficult than the Goldbach Conjecture. Our analysis shows that Collatz $3x + 1$ Problem, in its original formulation, is finitely refutable, but our proof *is not constructive*, so our method does not apply. However, two slightly stronger versions of this problem are finitely refutable, so they can be studied with our method.

## 2 THE METHOD

The Halting Problem for Turing machines is the problem to decide whether an arbitrary Turing machine $T$ eventually halts on an arbitrary input $x$. As a Turing machine $T$ can be coded by a finite string, say $code(T)$, it makes sense to ask whether we can design a Turing machine $T_{halt}$ which given $code(T)$ $x$, eventually stops and produces 1 if $T(x)$ stops and 0 if $T(x)$ does not stop. Turing's famous result states that this problem cannot be solved by any Turing machine, i.e. there is no such $T_{halt}$.

[2] At that time 1 was considered prime.

It is perhaps surprising to note that many problems in mathematics can be reformulated in terms of the halting/non-halting status of appropriately constructed Turing machines. For example, consider Fermat's Theorem, stating that there is no integers $x, y, z, n > 3$ such that $x^n + y^n = z^n$. It is not difficult to see how to construct a Turing machine $T_{Fermat}$ which enumerates systematically all possible integers (for example, written in binary) $x, y, z, n > 3$, checks whether $x^n + y^n = z^n$, and stops if for some values $x, y, z, n$ the relation is true (which would mean that the program has found a counter-example); otherwise, $T$ generates a new 4-tuple $x, y, z, n$ and repeats the above procedure. Clearly, Fermat's Theorem is equivalent with the statement "$T_{Fermat}$ never halts", hence knowing that Fermat's Theorem is true we know that $T_{Fermat}$ never halts. A possible (admittedly, not very promising) way to prove Fermat's Theorem is to show that $T_{Fermat}$ never halts. But this is not the point! The point is that we can measure the difficulty of Fermat's Theorem by the complexity of $T_{Fermat}$, for example, by the number of bits necessary to specify $T_{Fermat}$ in some fixed formalism. Of course, there are many Turing machines equivalent to $T_{Fermat}$, so a natural way to evaluate the complexity is to consider the least complex such machine.

In what follows we work with self-delimiting Turing machines $C$: such a machine reads the program's data (in binary) from right to left only, never going back, so its accepted programs (i.e., its domain) form a prefix-free set. If, after finitely many steps, $C$ halts with the program tape head scanning the last bit of the input $x$, then the computation is a success, and we write $C(x) < \infty$; the output of the computation is the binary string $C(x)$. Otherwise, the computation is a failure, we write $C(x) = \infty$, and there is no output.

A classical result states the existence of a universal self-delimiting Turing machine $U$: for every self-delimiting Turing machine $C$ there is a string $p$ (depending upon $U, C$) such that $U(px) = C(x)$, for all strings $x$. The halting probability of $U$, denoted by $\Omega_U$ (and called Chaitin's Omega number) is defined by $\Omega_U = \sum_{U(x)<\infty} 2^{-|x|}$, where $|x|$ denotes the length of $x$. With the first $N$ digits of $\Omega_U$ we can solve the Halting Problem for all programs $x$ of length less than or equal to $N$.

Consider now a problem $\Pi$ for which we can construct a program $C_\Pi$ such that $\Pi$ is false iff $U(C_\Pi) < \infty$ (if such a program exists). We propose to define the "difficulty" of a problem $\Pi$ by the minimal number of bits of $\Omega_U$ necessary to test whether $C_\Pi$ stops on $U$.

Many problems, including Riemann Hypothesis and Goldbach's Conjecture, are finitely refutable, so they can be classified according to our measure. Of course, not every problem is finitely refutable; an example is the Twin Prime Conjecture (TPC) (see [7], p. 337):

$$\forall n \{\exists p [p > n \,\&\, p \text{ prime } \&\, p + 2 \text{ prime}]\}.$$

However, the stronger Twin Prime Conjecture is finitely refutable:

$$\forall n \{\exists p [n < p < 2^{n+4} \,\&\, p \text{ prime } \&\, p+2 \text{ prime}]\}.$$

If the program corresponding to the strong TPC never halts, then the TPC is true; however, if the program does not halt, then we get no information about the TPC.

Theoretically, the choice of $U$ is irrelevant up to an additive constant, so if a problem is significantly more complex than another one with respect to a fixed universal self-delimiting machine, then it will continue to be more complex for any other machine. However, the proposed measure is *uncomputable* (see [2]), so in practice we just obtain an upper bound on the number of digits of $\Omega_U$ necessary to check whether $U(C_\Pi) < \infty$.

In practice, to evaluate the complexity of $\Pi$ we need to obtain effectively the program $C_\Pi$, and to compute its size in bits: this gives an upper bound on its complexity of $C_\Pi$, hence the difficulty of $\Pi$.

## 3 A UNIVERSAL SELF-DELIMITING TURING MACHINE

In what follows we will briefly describe the syntax and the semantics of a register machine program which implements a (natural) universal self-delimiting Turing machine; it is a refinement of the languages described in [3, 5].

Any register machine has a finite number of registers, each of which may contain an arbitrarily large non-negative binary integer. The list of instructions is given below in two forms: our compact form and its corresponding Chaitin style version. The main difference between Chaitin's implementation and ours is in the encoding: we use 4 bits instead of 8 bits per character.

By default, all registers, labeled with a string of 'a' to 'h' characters, are initialized to 0. It is a syntax error if the first occurrence of register $j$ appears before register $i$ in a program, where $j$ is lexicographic greater than $i$. Also, all registers lexicographic less than $j$ must have occurred. Instructions are labelled by default with 0,1,2,... (in binary).

The register machine instructions are listed below. Note that in all cases R2 denotes either a register or a binary constant of the form $1(0+1)^*+0$, while R1 and R3 must be register variables.

**=R1,R2,R3** **(EQ R1 R2 R3)**

If the contents of R1 and R2 are equal, then the execution continues at the R3-th instruction, where R3 = 0 denotes the first instruction of the program. If they are not equal, then execution continues with the next instruction in sequence. If the content of R3 is outside the scope of the program, then we have an illegal branch error.

**&R1,R2** **(SET R1 R2)**

The contents of register R1 is replaced by the contents of register R2.

**+R1,R2** **(ADD R1 R2)**

The contents of register R1 is replaced by the sum of the contents of registers R1 and R2.

**!R1** **(READ R1)**

One bit is read into the register R1, so the numerical value of R1 becomes either 0 or 1. Any attempt to read past the last data-bit results in a run-time error.

**%** **(HALT)**

This is the last instruction for each register machine program before the raw data. It halts the execution in two possible states: either successfully halts or it halts with an under-read error.

A *register machine program* consists of a finite list of labelled instructions from the above list, with the restriction that the `HALT` instruction appears only once, as the last instruction of the list. The input data (a binary string) follows immediately after the `HALT` instruction. A program not reading the whole data or attempting to read past the last data-bit results in a run-time error. Some programs (as the ones presented in this paper) have no input data.

To aid the presentation and development of the programs we use a consistent style for subroutines. We use the following conventions:

1 The letter 'L' followed by characters (usually $1, \ldots, 9$) and terminated by ':' is used to mark line numbers. These are local within the subroutine. References to them are replaced with the binary constant in the final program.
2 For unary subroutines, registers $a =$ argument, $b =$ return line, $c =$ answer ($a$ and $b$ are unchanged on return).
3 For binary subroutines, registers $a =$ argument1, $b =$ argument2, $c =$ return line, $d =$ answer ($a$, $b$ and $c$ are unchanged on return).
4 For subroutines, registers $e, \ldots, h$ are used for temporary values and may be modified. Also, up to register 'ah' should be considered reserved when using secondary level subroutines like `isPrime` and `Explog`.
5 For Boolean data types we use integers $0 =$ `false` and $1 =$ `true`.

## 4 GOLDBACH'S CONJECTURE

Goldbach's Conjecture, which is part of Hilbert's eighth problem, states that all positive even integers greater than two can be expressed as the

sum of two primes. The conjecture was tested up to $3 \times 10^{17}$ by Oliveira e Silva, [19].

The associated program is straightforward: we just enumerate all positive even integers greater than two and for each we check the required property (which is computable); the program stops if and only if it finds a counter-example for Goldbach's Conjecture. The program (in human readable form) is the following:

```
&a,Goldbach
=b,c,a  // b=c=0 so jumps to start of Goldbach algorithm

// Cmp(a,b) returns 1 if a<b, 0 if a=b, or 2 if a>b
&d,0
=a,b,c
+d,1
&e,a
&f,b
&g,L1
&h,L2
L1: +e,1
+f,1
=e,b,c
=f,a,h
=a,a,g
L2: +d,1
=a,a,c

// Sub(a,b) returns a-b, where we assume a>=b
&d,0
=a,b,c
&e,b
&f,L1
L1: +d,1
+e,1
=a,e,c
=a,a,f

// Mul(a,b) returns a*b
&d,0
=a,0,c
=b,0,c
&e,L1
&f,1
+d,a
```

```
L1: =f,b,c
+f,1
+d,a
=a,a,e

// Div(a,b) returns integer floor of a/b, assumes b>0
&g,L1
&h,L2
&aa,a     // copy of a (variable not used in Cmp & Mul)
&ab,b     // copy of b
&ac,c     // copy of c
&ad,1     // initial answer
&c,L0
&d,Cmp
=a,a,d    // call Cmp(a,b)
L0: &c,aa
=d,0,g
=d,2,h
&d,0      // else a < b so return 0
=a,a,c
L1: &d,1  // a=b so return 1
=a,a,c
L2: &c,L5
&ae,Mul
&a,ad
=a,a,ae
L5: &a,d  // just computed ad*b
&b,aa
&c,L6
&ae,Cmp
=a,a,ae   // call Cmp(ad*b,a)
L6: &b,ab // reset b
&ae,L3
=d,1,ae
&ae,L4
=a,a,ae
L3: +ad,1 // still <
=a,a,h
L4: &d,ad
&a,aa     // unpop input parameters
&b,ab
&c,ac
=a,a,c
```

```
// Mod(a,b) returns a mod b, assumes b>0
&af,a     // copy of parameters
&ag,b
&c,L0
&d,Div
=a,a,d    // call Div(a,b)
L0: &a,d
&c,L1
&d,Mul
=a,a,d    // call Mul(a/b,b)
L1: &a,af
&b,d
&c,L2
&d,Sub
=a,a,d    // call Sub(a,[a/b]*b)
L2: &b,ag
=a,a,c

// isPrime(a) where a >= 2
&d,0
&ah,a     // parameter copy
&ba,L1
&bb,L2
&bc,c
&b,2
L1: =b,ah,bb // trial reached so a is prime
&bd,Mod
&c,L3
=a,a,bd  //  call Mod(a,b)
L3: &c,bc
=d,0,c   // divisible by b so not prime
+b,1
=a,a,ba
L2: &d,1
&a,ah
=a,a,c

// start of Goldbach Conjecture testing program
&be,2
L0: +be,2 // enumeration of all even integers >= 4
&bf,2    // enumeration of all primes bf >= 2 and bf < be
&a,bf
&c,L1
```

```
&bg,isPrime
=a,a,bg
L1: &e,L3
=d,1,e     // if bf is prime
L2:+bf,1   // else increment
&d,L6
=bf,be,d   // found counter example
&a,bf
&c,L1
=a,a,bg    // check is prime again
L3: &bh,bf // enumeration of all primes bh >= bf and bh < be
L5: &ca,bf
+ca,bh     // sum of primes
&cb,L0
=be,ca,cb  // found sum, test next be
L7: +bh,1
&cb,L2
=bh,be,cb  // try next bf
&c,L4
&a,bh
=a,a,bg
L4: &e,L5
=d,1,e     // if bh is prime
&e,L7
=a,a,e     // else increment bh
L6: %
```

The machine executable version (see the Appendix) has 135 instructions totaling length 871 4-bit characters, i.e. its size in bits is 3484.

## 5 RIEMANN HYPOTHESIS

Riemann Hypothesis is a famous conjecture included in Hilbert's eighth problem: this is the hypothesis that the non-trivial complex zeros of Riemann's zeta function, which is defined for $Re(s) > 1$ by

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

lie exactly on the line $Re(s) = 1/2$.

According to Matiyasevich [15], p. 119–121, the negation of the Riemann hypothesis is equivalent to the existence of integers $k, l, m, n$ satisfying the following six conditions (here $x \mid z$ means "$x$ divides $z$"):

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

where $explog(a, b)$ denotes the predicate

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

The program is fairly straightforward, except for dealing with $explog(a, b)$, which is computably enumerable only; in this case we have adopted a dovetailing approach. Indeed, we just enumerate all 5-tuples $k, l, m, n, T$ satisfying conditions 1-3, 6 and the relativized conditions 4, 5:

4' $explog(m-1, l, T)$,
5' $explog(n-1, k, T)$,

where $explog(a, b, T)$ denotes the predicate

$$\exists x\,[T > x > b+1 \,\&\, (1+1/x)^{xb} \leq a+1 < 4(1+1/x)^{xb}].$$

In what follows we will present the program in human readable form only; the machine executable version can be obtained immediately (as in the case of Goldbach's program).

```
&a,Riemann
=b,c,a   // b=c=0 so jumps to start of Riemann algorithm

// Cmp(a,b) returns 1 if a<b, 0 if a=b, or 2 if a>b
// Sub(a,b) returns a-b, where we assume a>=b
// Mul(a,b) returns a*b
// Div(a,b) returns integer floor of a/b, assumes b>0
// Mod(a,b) returns a mod b, assumes b>0

// Pow(a,b) returns a^b where we assume a>0 and b>0
&d,a
=b,1,c   // a^1 = a base case?
&g,a
&h,b     // keep our own runtime stack for a,b,c
&aa,c
&ab,1
&ac,L1
&ad,L2
&c,L3
&b,a     // get ready to compute a*a
```

```
L2: +ab,1
&ae,Mul   // replace with address of subroutine Mul
=a,a,ae
L3: &a,d
=ab,h,ac  // stop when a^b is computed
=a,a,ad
L1: &a,g
&b,h      // unpop stack
&c,aa
=a,a,c

// Explog(a+1,b); uses global variable ah as upper bound
// to test if there exists ah >= x > b+1 such that
// (x+1)^{xb} <= (a+1)x^{xb} < 4(x+1)^{xb}
&d,0
=ah,0,c   // return if no upper bound set
&ba,a     // store parameters locally
&bb,b
&bc,c
&bd,b     // start checking x at b+2
+bd,2
L1: &a,bd  // compute x+1 and x*b and store into be and bf
&be,bd
+be,1
&a,bd
&b,bb
&c,L2
&d,Mul
=a,a,d
L2: &bf,d
&a,be     // compute bg = (x+1)^{bf}
&b,bf
&c,L3
&d,Pow
=a,a,d
L3: &bg,d
&a,bd   // compute x^{bf} and this times a+1, yielding bh
&c,L4
&d,Pow
=a,a,d
L4: &a,d
&b,ba
&c,L5
```

```
&d,Mul
=a,a,d
L5: &bh,d
&a,bg        // test first <=
&b,bh
&c,L6
&d,Cmp
=a,a,d
L6: &a,L9
=d,2,a       // if false try next x
&a,4         // else compute 4*bg
&b,bg
&c,L7
&d,Mul
=a,a,d
L7: &a,bh    // test second <
&b,d
&c,L8
&d,Cmp
=a,a,d
L8: &a,L10
=d,1,a       // test if found x
L9: +bd,1    // next x
&d,0
&a,L10
=bd,ah,a     // exit if reach upper bound
&d,L1
=a,a,d
L10: &a,ba   // return
&b,bb
&c,bc
=a,a,c

&ca,599  // start of Riemann Conjecture testing program
L1: +ca,1
&cb,600
L2: &ah,0
L3: &cc,0
L4: &cd,0
L5: &b,cb    // compute ce=ca-cb-ah-cc-cd (=ca-b)
+b,ah
+b,cc
+b,cd
```

```
&a,ca
&d,Sub
&c,L6
=a,a,d
L6: &ce,d
&cf,0      // check Cond 1
&d,L7
L9: +cf,1  // next y
=cf,cb,d   // did we check all y<n (if yes, check Cond 2)
&a,cf
&b,cd
&c,L8
&d,Mod
=a,a,d
L8: &cg,L9
=d,0,cg    // divides when a mod b is zero
&d,L10
=a,a,d
L7: &cf,1  // check Cond 2
LA2: &d,LA1
+cf,1      // next y
=cf,cd,d   // did we check all y<m (if yes, check Cond 3)
&be,0
&d,L10
=cd,0,d    // check m>0 (kludge!)
LA4: =be,n,d
+be,1      // did we try all x < n?  (next m, if so)
&a,be
&b,cf
&c,LA3
&d,Mod
=a,a,d
LA3: &c,LA4
=d,0,c     // divides so try next x
&d,LA2
=a,a,d     // do next y
LA1: &a,cd // check Cond 3
&b,cc
&c,LA5
&d,Explog
=a,a,d
LA5: &c,L10
=d,0,c     // if fail do next m
```

```
&a,cb       // check Cond 4
&b,ce
&c,LA6
&d,Explog
=a,a,d
LA6: &c,L10
=d,0,c      // if fail do next m
&a,cb       // check Cond 5
&b,ce
&c,LA9
&d,Mul
=a,a,d
LA9: &bg,d
+bg,bg      // bg=2nk
&a,cc
&b,cb
&c,LA7
&d,Cmp
a,a,d
LA7: &c,LA8 // if l > n
=d,2,c
&a,cb       // else l <= n
&b,bg
+b,cc
&c,LB0
&d,Cmp
a,a,d
LB0: &c,LX
=d,2,c      // last condition passes
&c,L10      // else try next m
=a,a,c
LA8: &a,cc
&b,bg
+b,cb
&c,LB1
&d,Cmp
a,a,d
LB1: &c,LX
=d,2,c      // 2nd way last condition passes
L10: &bg,cd // next m
+bg,cb
+bg,ah
+bg,cc
```

```
&d,L11
=bg,ca,d   // check at end of m loop
+cd,1       // else increment
&d,L5
=a,a,d
L11: &bg,cc // next l
+bg,cb
+bg,ah
&d,L12
=bg,ca,d   // check at end of l loop
+cc,1
&d,L4
=a,a,d
L12: &bg,ah // next b
+bg,cb
&d,L13
=bg,ca,d   // check at end of b loop
+ah,1
&d,L3
=a,a,d
L13: &d,L1  // next n and next s
+cb,1
=cb,ca,d
&d,L2
=a,a,d
LX: %
```

The register machine version of our Riemann hypothisis test consists of 290 lines and 1953 characters (7780 bits), as an upper bound on the complexity of the problem.

## 6 COLLATZ $3x + 1$ PROBLEM

Recall that Collatz function is defined by $T(x) = x/2$, if $x$ is even, and $T(x) = 3x + 1$, if $x$ is odd. The conjecture states that for every seed $a > 0$, there is an iteration $N$ such that $T^N(a) = 1$, see [14]. Again, this conjecture was tested and proved true up to $7 \cdot 2^{58}$, see Oliveira e Silva, [20].

First we note that a brute-force tester, i.e. the program which will enumerate all seeds and for each of them will try to find an iteration equal to 1, may never stop in two different cases: a) because the conjecture is indeed true, b) because for some specific seed $a_0$ there is no $N$ such that $T^N(a_0) = 1$. It is not clear how to differentiate these cases; even worse, it is not clear how to refute b) by a brute-force

tester.

Secondly, there is a *non-constructive* way to prove that there exists a program $C_{Collatz}$ which never stops if and only if the conjecture is true. Indeed, observe first that the set

$$Collatz = \{a > 0 \mid T^N(a) = 1, \text{ for some } N \geq 1\}$$

is a computably enumerable set. Collatz $3x + 1$ Problem requires to prove that $Collatz$ contains indeed all positive integers.

If $Collatz$ is not computable, then the conjecture is false, and this can be proven just by running the brute-force tester (which, we know, it will eventually produce a counter-example because a) is ruled out). If $Collatz$ is computable, then we can write a program to check whether each integer is in $Collatz$, so we can decide whether the conjecture is true or not. Of course, this argument has no practical value, because we don't know which of the above programs should be chosen.

We examine two possibilities to attack the Collatz $3x + 1$ Problem with our method. In the first one we search for a repeating cycle (the other possible case for finding a counter-example is when there is an infinite sequence of distinct integers). Our first program looks for a seed that begins such a cycle. This can be observed if the following statement is found to be false:

$$\forall a \geq 5 \, \forall N > 0 \, [T^N(a) \neq a].$$

We set the starting position at $a \geq 5$ since the repeating sequence of numbers $1, T(1) = 4, T^2(1) = T(4) = 2, T^3(1) = T(2) = 1, \ldots$ is not a counter-example. The main program for our Collatz one-sided tester, given below, systematically increases the maximum number of iterations to try (register `bb`) as it expands the set of integers (register `bc`) to test. If this program stops, then it disproves the original Collatz $3x + 1$ Problem by halting on a counter-example.

```
// start of Collatz loop checking program
// Cmp(a,b) returns 1 if a<b, 0 if a=b, or 2 if a>b
// Sub(a,b) returns a-b, where we assume a>=b
// Mul(a,b) returns a*b
// Div(a,b) returns integer floor of a/b, assumes b>0
// Mod(a,b) returns a mod b, assumes b>0

&ah,Mod
&ba,5   // integer a to try
&bb,1   // number of iterations
&bc,2   // current maxtry
L0: +ba,1
```

```
=ba,bc,L1
&bd,ba
&be,1
L2: =be,bb,L0
&a,bd
&b,2
&c,L3
a,a,ah      // call Mod(a,2)
L3: &c,L4
=d,0,c      // if even
&b,3        // else odd
&c,L5
&d,Mul
=a,a,d      // compute a*3
L5: +d,1    // compute a*3+1
&bd,L7
=a,a,bd
L4: &c,L7
&d,Div
=a,a,d      // compute a/2  (note b=2 from Mod call)
L7: &bd,d
&d,L6
=bd,ba,d    // if found a counter-example halt
&d,L0
=bd,1,d     // if we reach 1 then try next a
+be,1       // otherwise do another iteration
&d,L2
a,a,d
L1: +bc,1
+bb,1
&ba,1
=a,a,L0
L6: %
```

The final program has 127 lines of 808 characters, which totals a length of 3232 bits.

The second approach solves a time-bounded version of the original problem, namely:

$$\forall a \geq 2\, \exists N\, [N \leq 2^a \,\&\, T^N(a) = 1].$$

If the corresponding program never stops, then it will prove the original Collatz $3x + 1$ Problem. Here is the program:

```
// start of Collatz time-bounded checking program
```

```
// Cmp(a,b) returns 1 if a<b, 0 if a=b, or 2 if a>b
// Sub(a,b) returns a-b, where we assume a>=b
// Mul(a,b) returns a*b
// Div(a,b) returns integer floor of a/b, assumes b>0
// Mod(a,b) returns a mod b, assumes b>0
// Pow(a,b) returns a^b where we assume a>0 and b>0


&ah,1       // integer a to try
L7: +ah,1
L0: &a,2
&ba,ah      // current value of T*(a)
&b,ba
&c,L1
&d,Pow
=a,a,d      // compute bound 2^ a
L1: &bb,d
&bc,0       // iteration counter
L2: +bc,1
&a,ba       // else compute next T()
&b,2
&c,L3
&d,Mod
a,a,d       // call Mod(a,2)
L3: &c,L4
=d,0,c      // if even
&b,3        // else odd
&c,L5
&d,Mul
=a,a,d      // compute a*3
L5: +d,1    // compute a*3+1
&c,L6
=a,a,c
L4: &c,L6   // even case
&d,Div
=a,a,d      // compute a/2  (note b=2 from Mod call)
L6: &ba,d
&d,L7
=ba,1,d     // if passes test then try next a
&d,LX       // while less than bound
=bc,bb,d    // if not, found counter-example
&d,L2
=a,a,d      // else next iteration of Collatz on a
LX: %
```

The final program has 143 lines of 897 characters, which totals a length of 3588 bits.

## 7 FINAL COMMENTS

Our analysis gives a new method of comparing the difficulties of two or more finitely refutable problem. The main obstacle is the non-computability of the measure. However, working with upper bounds one can obtain a practical method for evaluating the complexity. Using this method we showed that the Riemann Hypothesis is more difficult than the Goldbach Conjecture. Two stronger versions of Collatz $3x + 1$ Problem have been also studied.

Trying to reduce the lengths of our programs should be possible; of course, proving minimality is much more demanding.

## ACKNOWLEDGMENT

We thank John Tromp for comments which improved our presentation.

## APPENDIX

Here is the final machine executable version of the Goldbach program, where one reads the sequence of instructions down each of the three columns. 3

```
&a,1101000
=b,c,a
&d,0
=a,b,c
+d,1
&e,a
&f,b
&g,1001
&h,1110
+e,1
+f,1
=e,b,c
=f,a,h
=a,a,g
+d,1
=a,a,c
&d,0
```

```
=a,b,c
&e,b
&f,10100
+d,1
+e,1
=a,e,c
=a,a,f
&d,0
=a,0,c
=b,0,c
&e,11110
&f,1
+d,a
=f,b,c
+f,1
+d,a
=a,a,e
&g,110000
&h,110010
&aa,a
&ab,b
&ac,c
&ad,1
&c,101011
&d,10
=a,a,d
&c,aa
=d,0,g
=d,10,h
&d,0
=a,a,c
&d,1
=a,a,c
&c,110110
&ae,11000
&a,ad
=a,a,ae
```

3

```
&a,d
&b,aa
&c,111011
&ae,10
```

```
=a,a,ae
&b,bb
&ae,1000000
=d,1,ae
&ae,1000010
=a,a,ae
+ad,1
=a,a,h
&d,ad
&a,aa
&b,ab
&c,ac
=a,a,c
&af,a
&ag,b
&c,1001100
&d,100010
=a,a,d
&a,d
&c,1010000
&d,11000
=a,a,d
&a,af
&b,d
&c,1010101
&d,10000
=a,a,d
&b,ag
=a,a,c
&d,0
&ah,a
&ba,1011101
&bb,1100110
&bc,c
&b,10
=bd,ah,bb
&bd,1000111
&c,1100001
=a,a,bd
&c,bc
=d,0,c
+b,1
=a,a,ba
```

```
&d,1
&a,ah
=a,a,c
&be,10
+be,10
&bf,10
&a,bf
&c,1101111
&bg,1010111
=a,a,bg
&e,1110111
=d,1,e
+bf,1
&d,10000110
=bf,be,d
&a,bf
&c,1101111
=a,a,bg
&bh,bf
&ca,bf
+ca,bh
&cb,1101001
=be,ca,cb
+bh,1
&cb,1110001
=bh,be,cb
&c,10000010
&a,bh
=a,a,bg
&e,1111000
=d,1,e
&e,10000100
=a,a,e
%
```

## REFERENCES

[1] Browder, F.E. (ed.). (1976). *Mathematical Developments Arising from Hilbert Problems,* Amer. Math. Soc., Providence, RI.

[2] Calude, C.S. (2002). *Information and Randomness: An Algorithmic Perspective,* 2nd Edition, Revised and Extended, Springer-Verlag, Berlin.

[3] Calude, C.S., Dinneen, M.J. and Shu, C.-K. (2002). Computing a glimpse of randomness, *Experimental Mathematics 11,* 2 369–378.

[4] Calude, C.S., Jürgensen H. and Legg, S. (2000). Solving finitely refutable mathematical

problems, in Calude, C. S., Păun, G. (eds.). *Finite Versus Infinite. Contributions to an Eternal Dilemma,* Springer-Verlag, London, 39–52.

[5] Chaitin, G.J.. (1987). *Algorithmic Information Theory,* Cambridge University Press, Cambridge. (third printing 1990)

[6] Chaitin, G.J. (2004). Thoughts on the Riemann Hypothesis, *The Mathematical Intelligencer 26,*1 4–7.

[7] Davis, M., Matiyasevich, Yu. V. and Robinson, J. Hilbert's tenth problem: Positive aspects of a negative solution, in [1], 323–378.

[8] Dehn, M. (1900). Über raumgleiche Polyeder, *Nachr. Königl. Ges. der Wiss, zu Göttingen f.d. Jahr,* 345–354.

[9] Derbyshire, J. (2003). *Prime Obsession,* Joseph Henry Press.

[10] Dickson, L.E. (2005). *Goldbach's Empirical Theorem: Every integer is a sum of two primes,* in L. E. Dickson. *History of the Theory of Numbers, 1, Divisibility and Primality,* Dover, New York, 421–424.

[11] Goldbach, C. (1742). Letter to L. Euler, http://www.mathstat.dal.ca/˜joerg/pic/g-letter.jpg.

[12] Havil, J. (2003). *Gamma*, Princeton University Press.

[13] Hilbert, D. Mathematical Problems, (1901–1902) *Bull. Amer. Math. Soc. 8,* 437–479.

[14] Lagarias, J.C. (1985). The $3x + 1$ problem and its generalizations, *Amer. Math. Monthly 92,* 3–23.

[15] Matiyasevich, Yu. V. (1993). *Hilbert's Tenth Problem,* MIT Press, Cambridge, MA.

[16] Odifreddi, P. (2004). *The Mathematical Century: The 30 Greatest Problems of the Last 100 Years,* Princeton University Press.

[17] Sabbagh, K. (2003). *The Riemann Hypothesis,* Farrar, Strauss and Giroux.

[18] du Sautoy, M. (2003). *The Music of the Primes,* Harper Collins.

[19] Oliveira e Silva, T. (2005). Goldbach Conjecture verification, http://www.ieeta.pt/˜tos/goldbach.html.

[20] Oliveira e Silva, T. (2005). Computational verification of the 3x+1 conjecture, http://www.ieeta.pt/˜tos/.

[21] Thiele, R. (2001). On Hilbert's 24th Problem: Report on a New Source and Some Remarks, *AMS/MAA Joint Mathematics Meeting*, New Orleans.

[22] http://www.claymath.org/millennium/Riemann_Hypothesis/.