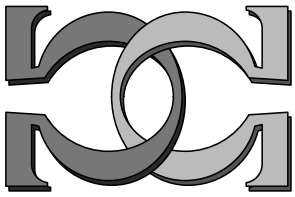

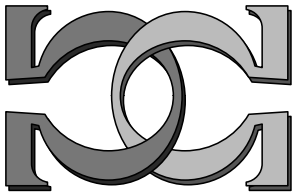

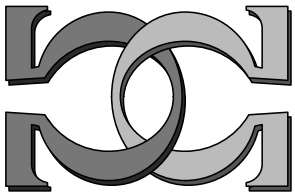

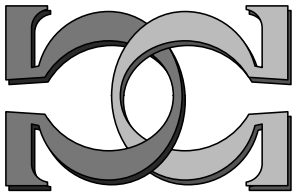

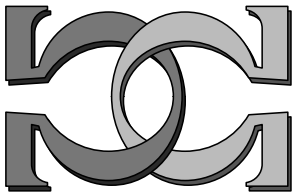

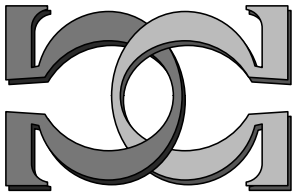

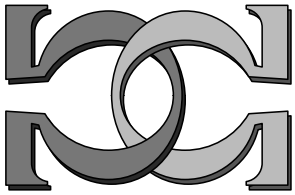

CDMTCS Research Report Series

On the Difficulty of Goldbach and Dyson Conjectures

Joachim Hertel
H-Star, USA

CDMTCS-367
July 2009

Centre for Discrete Mathematics and
Theoretical Computer Science

ON THE DIFFICULTY OF GOLDBACH AND DYSON CONJECTURES

JOACHIM HERTEL

Abstract. Using a Measure of Difficulty recently presented by Calude et al. in [1], we show that Goldbach's Conjecture is less difficult than previously assumed. Dyson's Conjecture states that the reverse of a power of 2 cannot be written as a power of five and might serve as an easy to understand example of a true but unprovable statement.We verify Dyson's Conjecture for all 2^k with $k \leq 10^8$ and show that it's complexity has an upper bound of 3928 bits. We conclude that the Measure of Difficulty for Dyson's Conjecture is roughly comparable to that of a strong version of the Collatz 3x+1 problem.

1. Introduction

Some conjectures in Mathematics can be stated in elementary terms only. Almost 270 years ago, Goldbach wrote a letter [6] to Euler, in which he formulated what is now known as the Goldbach Conjecture (GC).

Conjecture 1 (GC)**.** *Every even integer greater than 2 is the sum of two primes.*

More recently, Freeman Dyson contributed to the EDGE topic *what you believe but cannot prove* a mathematically minded answer [5]. Dyson presents a kind of recipe for constructing (probably) true, but unprovable statements. Since Goedel we know, that there are true mathematical statements that cannot be proved. Dyson gives one embodiment for his recipe, now known as the Dyson Conjecture (DC).

Conjecture 2 (DC)**.** *The reverse of a power of two is not a power of five.*

Neither conjecture is proven yet. Dyson even argues, that DC might be true but unprovable, due to the (yet unproven) fact, that the digits in large powers of two occur randomly. In [3] Calude comments on that and verifies DC for all 2^k with $k \leq 10^5$.

How difficult are these easy to understand and yet hard to prove or even unprovable conjectures? In [1] Calude et al. proposed a Measure of Difficulty based on the well known fact [2] that many problems in Mathematics, including GC and DC, can be re-stated in terms of the halting question for a suitable Turing Machine. One can then measure the difficulty of the problem under consideration as the complexity of the associated Turing Machine. In [1] Calude et al. devised a method to do so, by exhibiting a set of instructions and semantic rules for register machines that provide a natural implementation of a self delimiting, universal Turing Machine. Both conjectures,GC and DC, belong to the class of finitely refutable problems, that is, one counter example proves the conjecture to be false. Hence one constructs the

Date: July 19, 2009.

register machine programs for these problems such that they HALT iff a counter example is found. However, as long as the program does not HALT, we are not gaining any knowledge about whether the conjecture is true or false. We only learn, that the conjecture is valid for ever increasing intervals. Using this method, Calude et al. showed that the complexity of GC has an upper bound of 3484 bits. In chapter 3 we improve this upper bound to 1628 bits. In chapter 4 we verify DC for all 2^k with $k \leq 10^8$ and show that the complexity of DC has an upper bound of 3928 bits. To be self contained, we recall from [1] the syntax and semantics of register machine programs.

2. A Universal Self-Delimiting Turing Machine

In [1] Calude et al. present the syntax and semantics of a register machine that implements a universal self-delimiting Turing Machine. We refer to [2] for the background on Algorithmic Information Theory.

Any register machine has a finite number of registers, each of which may contain an arbitrarily large non-negative binary integer. For encoding we use 4 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.

$=\mathbf{R1},\mathbf{R2},\mathbf{R3}$: 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.

$\&\mathbf{R1},\mathbf{R2}$: The contents of register R1 is replaced by the contents of register R2.

$+\mathbf{R1},\mathbf{R2}$: The contents of register R1 is replaced by the sum of the contents of registers R1 and R2.

!R1 : One bit is read into the register R1, so the numerical value of R1 becomes either 0or 1. Any attempt to read past the last data-bit results in a run-time error. Note: Read instructions are not used in GC nor DC.

%: This is the HALT and 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 0, . . . , 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, . . . , h are used for temporary values and may be modified.
(5) For Boolean data types we use integers 0 = False and 1 = True.

3. An Improved Upper Bound for the Complexity of the Goldbach Conjecture

In [1] Calude et al. present an upper bound for the complexity of the Goldbach Conjecture. They show $Complexity(GC) \leq 3848\ bits$. We improve this upper bound to 1628 bits[1].

The improvement is largely based on a simple insight which saves many register machine program instructions and thus reduces the complexity. Given two integers a and b, the register machine program in [1] relies on the argument that b does not divide a if $Mod(a, b) = 0$. That is correct. However, the implementation of the Mod function consumes many instructions. We devise a simple boolean binary subroutine $isDivisible(a, b)$ that returns True if b is a (proper) divisor of a and False otherwise. This subroutine needs 11 instructions only. We also use relative addressing to streamline the subroutine $isPrime()$ and the main routine *Goldbach* as displayed in [1]. By relative addressing we mean this. Assume one needs to load labels L1 and $L2 > L1$ to registers c and d, respectively. Usually one uses the instructions $\&c, L1$ and $\&d, L2$. An alternative way to achieve this is to use $\&c, L1$ followed by $\&d, c$ and $+d, k$, with $k = L2 - L1$. A quick calculation shows, that relative addressing consumes less characters as long as

$$Lenght[L2] - Length[k] \geq 4$$

We consequently apply this recipe throughout the code. The annotated register machine program to test GC is as follows:

[1]Cristian S. Calude made available a draft version dated 7/19/2009 of related work [4] in which the complexity of GC is even further reduced to 1068 bits.

```
&a, Goldbach        //load Adr(Goldbach)
= b, c, a           //goto Goldbach

//isDivisible(a, b) //returns True if b is a proper divisor of a ,else False.1 < b < a is assumed
&d, 0
&e, L0
&f, b
&g, 1
L0 : +f, b
+g, 1
&d, 0
= a, g, c           //return False, if g = a
&d, 1
= a, f, c           //return True, if a = g * b with g integer and g < a
+a, a, e

//isPrime(a)        //returns True if a is prime, else False
&h, b               //save return address
&aa, L0             //load Adr(L0)
&ab, L1             //load Adr(L1)
&b, 10              // b will be a potential proper divisor of a . Init b with 2
L4 := a, b, ab      //exit to L1 if b has reached a: we have not found a proper divisor,hence a is prime
&d, isDivisible     //load Adr(isDivisible)
&c, L3
= a, a, d           //perform isDivisible(a, b)
L3 := d, 1, aa      //yes, 1 < b < a is a proper divisor of a,hence a is NOT prime,goto L0
+b, 1               //continue testing with the successor of b
&d, L4
= a, a, d           //continue at L4
L0 : &c, 0          //prepare to exit with False, i.e. a is not prime
&b, h               //reload return address
= a, a, b           //go back to caller
L1 : &c, 1          //prepare to exit with True, i.e. a is prime
&b, h               //reload return address
= a, a, b           //go back to caller
```

//Goldbach	**start of Goldbach Conjecture testing program**
$\&ac, 10$	//ac enumerates all even integers
$\&ad, isPrime$	//load $Adr(isPrime)$
$\&ae, L5$	//load $Adr(L5)$
$\&af, L0$	//load $Adr(L0)$
$\&ag, L2$	//load $Adr(L2)$
$\&b, L6$	//register b will always be the return address for isPrime()
$\&ah, b$	//set ah to $L7 = L6 + 7$
$+ah, 111$	//set ah to $L7 = L6 + 7$
$L0 : +ac, 10$	//start loop over all even integers greater than 2
$\&a, 10$	//start finding prime p_1
$\&ba, ae$	//switch branch labels to p_1 search mode
$\&bb, ag$	//switch branch labels to p_1 search mode
$= a, a, ad$	//perform $isPrime(a)$
$L6 := c, 1, bb$	//if yes go to L2
$L7 : +a, 1$	//test the successor of a
$= a, ac, ba$	// if $a = ac$: end p_2 search mode at L4, or end p_1 search mode at L5
$= a, a, ad$	//perform $isPrime(a)$
$L2 : \&bc, a$	//save prime p_1 to bc
$\&ba, L4$	//switch branch labels to p_2 search mode: set ba to L4
$+bb, \Delta$	//switch branch labels to p_2 search mode: set bb to $L3 = L2 + \Delta$
$L3 : \&bd, a$	//save a as prime p_2 in register bd
$+bd, bc$	//set bd to $p_1 + p_2$
$= ac, bd, af$	//test GC: if $ac = p_1 + p_2$ proceed with next even integer at L0
$= a, a, ah$	//$ac \neq p_1 + p_2$: continue at $L7$ in p_2 search mode as long as $p_1 \leq p_2 < ac$
$L4 : \&ba, ae$	//switch branch labels to p_1 search mode
$\&bb, ag$	//switch branch labels to p_1 search mode
$\&a, bc$	//reload p_1 back to register a
$= a, a, ah$	//continue testing at L7
$L5 : \%$	//HALT: counter example found: even integer ac is not the sum of two primes

The machine executable version of the Goldbach Conjecture testing program (see Appendix 2) complies with the syntax and semantic rules as explained in Chapter 2. It has 60 register machine instructions and consists of 407 4-bit characters. It has a complexity of 1628 bits. Thus we have reduced the upper bound for the complexity of the GC testing program from 3484 bits to 1628 bits and conclude, that the Goldbach Conjecture is less difficult than previously estimated.

Numerical Verification of the Dyson Conjecture

For $n \in \mathbb{N}$ let $\Re(n)$ denote the integer with the decimal digits of n in reverse order.

Example 1. $n = 256$ *gives* $\Re(256) = 652$

The Dyson Conjecture (DC) states, that

$$\forall k, j > 0, \Re(2^k) \neq 5^j$$

We have verified DC to be valid for $0 < k \leq 10^8$. We need to test only those exponents k that fulfill certain conditions that need to be true for a potential DC counterexample. These conditions allow a fast implementation and sieve out roughly 90% of integers in any large interval on $\mathbb{N}$. This is due to the probable fact, that digits in large powers of two occur randomly. In particular, on the interval $[1, 10^8]$ only $1,107,630$ integers allow a potential DC counterexample. All other exponents k give values of 2^k such that it is immediately clear that $\Re(2^k)$ cannot be a power of 5. First, we observe that k must be such that 2^k has the most significant decimal digit (MSD) 5. If $MSD(2^k) \neq 5$ then obviously $\mathrm{mod}(\Re(2^k), 5) \neq 0$ and hence $\Re(2^k)$ cannot be a power of 5. Next,assume k to be such that $MSD(2^k) = 5$. Let u denote the least significant decimal digit of 2^k. Obviously $u = \mathrm{mod}(2^k, 10)$ and since $u \in \{2, 4, 6, 8\}$ we conclude that 2^k and $\Re(2^k)$ must have the same number d of decimal digits, with d given by

$$d = floor(k * \log(10, 2)) + 1$$

and $MSD(\Re(2^k)) = u$. Now, to be a DC counterexample, 5^j must also have d decimal digits, and we conclude that the integer j is given by

$$j = floor(d * \log(5, 10)$$

For any DC counter example we further observe that $MSD(\Re(2^k)) = u = MSD(5^j)$. That further restricts j to be such that

$$\log(10, u) \leq fractionalpart(j * \log(10, 5)) < \log(10, u + 1)$$

We call an integer k **DC-admissable**, if k fulfills these criteria and allows for a suitable exponent j. We have generated[2] all DC-admissable integer $k \in [1, 10^8]$. In this interval, a total of $1,107,630$ DC-admissable integer exist. The smallest one is $k = 29$ with an allowed $j = 12$. However, testing all DC-admissable integers $k \in [1, 10^8]$ shows that all of them produce values for 2^k such that $\Re(2^k) \neq 5^j$. Hence the Dyson Conjecture is valid for $\forall k \in [1, 10^8]$.

The *Mathematica*$^{\mathrm{TM}}$ implementation for verifying DC is shown in Appendix 1.

4. The Difficulty of the Dyson Conjecture

The program to test DC is straightforward. We enumerate all powers of 2, and for each 2^k we compute the reverse integer $\Re(2^k)$.We then enumerate all powers of 5 up to $\Re(2^k)$ and check if there exists an integer j, such that $5^j = \Re(2^k)$.The program HALTS iff it finds a pair of integers $(k > 0, j > 0)$ such that $5^j = \Re(2^k)$.Otherwise it continues with the next power of 2. The register machine program uses subroutines Cmp,Sub,Mul,Div and Mod which where taken

[2]The data are available on request from the author

from [1]. Here, we present these subroutines in a re-written and streamlined form. The new subroutine Rev computes the reverse of a given integer. The main routine Dyson is the actual test program for the Dyson Conjecture. The annotated version is as follows:

&a,Dyson	**//Load Adr(Dyson)**
=b,c,a	//goto Dyson
//Cmp(a,b)	**Returns 1 if** $a < b$**, 0 if** $a = b$**, 2 if** $a > b$
$\&d, 0$	//Set return value to 0
$= a, b, c$	//return to caller if $a = b$
$\&e, 0$	
$\&f, L1$	
$\&g, L2$	
$\&d, 1$	//set return value to 1
$L1 := e, a, c$	//return to caller if $a < b$
$= e, b, g$	//go to L2 if $a > b$
$+e, 1$	
$= a, a, f$	
$L2 : +d, 1$	//set return value to 2
$= a, a, c$	//return to caller with $a > b$
//Mul(a,b)	**Returns** $a * b$
$\&d, 0$	//Set return value to 0
$\&e, L1$	
$\&f, 0$	
$L1 := f, b, c$	//if $f = b$ return to caller with $a * b$
$+f, 1$	
$+d, a$	
$= a, a, e$	
//Sub(a,b)	**Returns** $a - b$**,** $a \geq b$ **is assumed**
$\&d, 0$	
$= a, b, c$	
$\&e, b$	
$\&f, L1$	
$L1 : +d, 1$	
$+e, 1$	
$= a, e, c$	
$= a, a, f$	

//Div(a,b) **returns** $floor(\frac{a}{b})$**,** $b > 0$ **is assumed**

```
&d, 1
= a, b, c      //return 1 if a = b
&h, c          //save return address c to h
&aa, L1
&ab, b         //save input b to ab
&ac, Cmp       //load Adr(Cmp)
&ad, 0         //init counter to 0
&c, aa         //load Adr(L1)
= a, a, ac     //perform Cmp(a, b*(1+ad))
&c, L2         //load Adr(L2)
= d, 0, c      //go to c if floor(a/b) = 1+ad
+c, 101        //set c to Adr(L3) = Adr(L2)+5
= d, 1, c      //go to c if floor(a/b) = ad
+ad, 1         //continue
+b, ab         //continue
&c, aa         //reload c to Adr(L1)
= a, a, ac     //perform Cmp(a, b*(1+ad))
L2 : +ad, 1    //add deferred 1 to counter
L3 : &d, ad    //set return value
&b, ab         //reload input value
&c, h          //reload return address
= a, a, c      //return to caller
```

//Mod(a,b) **//returns** $a \bmod b, b > 0$ **is assumed**

```
&ae, a         //save input a
&af, b         //save input b
&ag, c         //save input c
&c, L1
&d, Div        //load Adr(Div)
= a, a, d      //perform Div(a, b)
L1 : &a, d
&d, Mul        //load Adr(Mul)
+c, Δ          //set c to Adr(L2) by adding Δ = Adr(L2) − Adr(L1)
= a, a, d      //perform Mul(floor(a/b), b)
L2 : &a, ae    //reload input a
&b, d
&d, Sub        //load Adr(Sub)
+c, Δ          //set c to Adr(L3) by adding Δ = Adr(L3) − Adr(L2)
= a, a, d      //perform Sub(a, b*floor(a/b))
L3 : &b, af    //reload input b
&c, ag         //reload return address
= a, a, c      //return to caller
```

//Rev(a,b)	**returns the reverse of** a; $b = 10^n$ **with** $n = floor(log(10, a)) + 1$ **is assumed**
$\&ah, c$	//save input
$\&ba, a$	//save input
$\&bb, b$	//save input
$\&bc, 1$	//counter bc keeps track of digit count and is 10^d with $d \geq 0$ the digit position
$\&bd, a$	//bd will hold the leading digits of a
$\&be, b$	// be contains 10^k with $0 \leq k \leq n = floor(log(10, a)) + 1$
$\&bf, 1010$	//load constant decimal 10
$\&bg, Mul$	//load $Adr(Mul)$
$\&bh, Div$	//load $Adr(Div)$
$\&ca, 0$	//ca keeps the reversed integer
$L0 : \&a, be$	//prepare to go from 10^k to 10^{k-1}
$\&b, bf$	
$\&c, L1$	
$= a, a, bh$	//perform $Div(10^k, 10)$
$L1 : \&be, d$	//update be from 10^k to 10^{k-1}
$\&b, d$	
$\&a, bd$	
$+c, 11101$	//set c to $Adr(L2)$ by adding $Adr(L2) - Adr(L1) = 29$
$= a, a, bh$	//perform $Div(a, 10^{k-1})$
$L2 : \&a, d$	//load a with the result: $a = d = floor(\frac{a}{10^{k-1}})$
$\&b, bc$	
$+c, 11000$	//set c to $Adr(L3)$ by adding $Adr(L2) - Adr(L1) = 24$
$= a, a, bg$	//perform $Mul(a, bc)$, where bc is 10^d with $d \geq 0$ the digit position
$L3 : +ca, d$	//add to cb, thus building the reversed integer
$\&a, bd$	
$\&b, be$	
$\&d, Mod$	//load $Adr(Mod)$
$+c, \Delta$	//set c to $Adr(L4)$ by adding $\Delta = Adr(L4) - Adr(L3)$
$= a, a, d$	//perform $Mod(be, bf)$, thus stripping off the current leading digit
$L4 : \&bd, d$	//update be with the leading digit removed
$\&a, bc$	//prepare the next round to multiply bd by 10 for the next digit position
$\&b, bf$	//load constant decimal 10
$+c, 11110$	//set c to $Adr(L5)$ by adding $Adr(L5) - Adr(L4) = 30$
$= a, a, bg$	//perform $Mul(bd, 10)$
$L5 : \&bc, d$	//update $bd = 10 * bd$
$\&a, ba$	//reload input a
$\&b, bb$	//reload input b
$\&c, ah$	//reload input c
$\&d, ca$	//load reversed integer
$= be, 1, c$	//if $be = 1$, we are done, that is all digits of a have been reversed,branch back to caller
$\&d, L0$	//$be \neq 1$, not all digits are yet reversed. Continue with next round at L0
$= a, a, d$	//branch to L0

//Dyson	**start of Dyson Conjecture testing program**
$\&cb, Cmp$	//load $Adr(Cmp)$
$\&cc, Mul$	//load $Adr(Mul)$
$\&cd, 1$	//init cd to enumerate 2^k
$L0 : +cd, cd$	// go from 2^k to 2^{k+1}
$\&ce, 1010$	//prepare the loop...
$\&cf, 1$	//...to determine smallest $cf = 10^n > cd$
$L1 : \&a, cf$	
$\&b, ce$	
$\&c, L2$	
$= a, a, cc$	//perform $Mul(cf, 10)$
$L2 : \&cf, d$	//update cf to $10 * cf$
$\&a, cf$	
$\&b, cd$	
$+c, 11110$	//set c to $L3 = L2 + 30$
$= a, a, cb$	//perform $Cmp(cf, cd)$
$L3 : \&c, L1$	
$= d, 1, c$	//goto L1 if $cf < cd$
$\&a, cd$	//here we have the smallest $cf = 10^n > cd$
$\&b, cf$	
$\&c, L4$	
$\&d, Rev$	//load Adr(Rev)
$= a, a, d$	//perform Rev(2^k)
$\&a, d$	
$\&b, 1$	
$+c, 10111$	//set c to $L5 = L4 + 23$
$= a, a, cb$	//perform Cmp(Rev(2^k), 5^j), for $j \geq 0$
$L5 : \&c, L0$	
$= d, 1, c$	//if Rev(2^k) $< 5^j$ goto next power of 2 at L0
$\&c, L6$	
$= d, 0, c$	//if Rev(2^k) $= 5^j$ goto L6
$\&g, a$	//Rev(2^k) $> 5^j$ proceed with next power of 5. Save Rev(2^k)
$\&a, 101$	//prepare to go to the next power of 5
$\&c, L7$	
$= a, a, cc$	//perform $Mul(5, 5^j)$ (Note: Mul is not using register g)
$L7 : \&b, d$	//load b with next power of 5
$\&a, g$	//reload a with Rev(2^k)
$\&c, L5$	//load return address for Cmp
$= a, a, cb$	//perform Cmp(Rev(2^k), 5^{j+1})
$L6 : \%$	//HALT: counter example found : $\exists k, j > 0$ such that Rev(2^k) $= 5^j$

The machine executable version of the Dyson Conjecture testing program (see Appendix 3) complies with the syntax and semantic rules as explained in Chapter 2. It has 150 register machine instructions and consists of 982 4-bit characters. It has a complexity of 3928 bits. Its complexity roughly compares to that of a strong version of the Collatz 3x+1 problem for which an upper bound of 3232 bits was given by Calude et al. in [1].

Acknowledgement. *The author thanks Cristian S. Calude for review and comments and for making available related research results prior to publication.*

5. Appendix 1

The Mathematica™ **program DCTest[k] returns k if k is DC-admissable, otherwise 0.**

$$LD[a\_Integer,\ b\_Integer,\ n\_Integer]\ :=\ Log10[n]\ <=\ FractionalPart[b*Log10[a]]\ <\ Log10[n\ +\ 1]$$

```
DCTest[k_Integer] := Module[{ret = 0, s = 2, j = 0, d = 1},
 (*check if 2^k has leading digit 5*)
 If[LD[2, k, 5],
 (*yes,now calculate the least significant digit s of 2^k*)
 s = PowerMod[2, k, 10];
 (*calculate the number of digits d of 2^k*)
 d = Floor[k*Log10[2]] + 1;
 (*calculate the possible index j for a possible 5^j having the same number of digits
and the correct leading digit*)
 j = Floor[d* Log[5, 10]];
 (*now check if leading digits of 5^j is the same as least significant digit of 2^k*)
 If[LD[5, j, s],ret = k,Null],
 Null
 ];
 Return[ret]
 ];
```

The Mathematica™ program **isDysonQ[n]** takes an integer **n** as input and returns True if an integer **j** exists such that $\mathbf{Reverse}[\mathbf{2^n}] = \mathbf{5^j}$, else False.

$$DCQ[n\_Integer] :=\ Log[5,\ FromDigits[Reverse[IntegerDigits[2\hat{}n]]]]$$

```
isDysonQ[n_Integer]:= Block[{p = DCQ[n]}, IntegerPart[p] == p];
```

Note on Performance:
Using Mathematica™ 7 on an Intel™ Core2™ Quad™ CPU with **2.66** GHz, WinXP SP3 and using the statement

$$\mathbf{Apply}[Or, \mathbf{ParallelMap}[isDysonQ, DC]]$$

with **DC** the set of $1,107,630$ DC-admissable integers in $[1, 10^8]$ needs roughly **950h** of elapsed time to produce **False**.

6. Appendix 2

The machine executable version of the Goldbach Conjecture testing program. Read the sequence of instructions down each of the two columns, left to right.

```
&a,11000000
=b,c,a
&d,0
&e,100110
&f,b
&g,1
+f,b
+g,1
&d,0
=a,g,c
&d,1
=a,f,c
=a,a,e
&h,b
&aa,10100100
&ab,10110010
&b,10
=a,b,ab
&d,10001
&c,10001001
=a,a,d
=d,1,aa
+b,1
&d,1101001
=a,a,d
&c,0
&b,h
=a,a,b
&c,1
&b,h
=a,a,b
&ac,10
&ad,1001000
&ae,110010110
&af,100010000
&ag,101001000
&b,100101110
&ah,b
+ah,111
+ac,10
&a,10
&ba,ae
&bb,ag
=a,a,ad
=c,1,bb
+a,1
=a,ac,ba
=a,a,ad
&bc,a
&ba,101111110
+bb,11011
&bd,a
+bd,bc
=ac,bd,af
=a,a,ah
&ba,ae
&bb,ag
&a,bc
=a,a,ah
%
```

7. Appendix 3

The machine executable version of the Dyson Conjecture testing program. Read the sequence of instructions down each of the four columns, left to right.

```
&a,1010111010
=b,c,a
&d,0
=a,b,c
&e,0
&f,111000
&g,1001110
&d,1
=e,a,c
=e,b,g
+e,1
=a,a,f
+d,1
=a,a,c
&d,0
&e,1101010
&f,0
=f,b,c
+f,1
+d,a
=a,a,e
&d,0
=a,b,c
&e,b
&f,10010111
+d,1
+e,1
=a,e,c
=a,a,f
&d,1
=a,b,c
&h,c
&aa,11100100
&ab,b
&ac,10011
&ad,0
&c,aa
=a,a,ac
&c,100011000
=d,0,c
+c,101
=d,1,c
+ad,1
+b,ab
&c,aa
=a,a,ac
+ad,1
&d,ad
&b,ab
&c,h
=a,a,c
&ae,a
&af,b
&ag,c
&c,101011101
&d,10101011
=a,a,d
&a,d
&d,1011000
+c,11100
=a,a,d
&a,ae
&b,d
&d,1111110
+c,100010
=a,a,d
&b,af
&c,ag
=a,a,c
&ah,c
&ba,a
&bb,b
&bc,1
&bd,a
&be,b
&bf,1010
&bg,1011000
&bh,10101011
&ca,0
&a,be
&b,bf
&c,1000001011
=a,a,bh
&be,d
&b,d
&a,bd
+c,11101
=a,a,bh
&a,d
&b,bc
+c,11000
=a,a,bg
+ca,d
&a,bd
&b,be
&d,100110001
+c,101010
=a,a,d
&bd,d
&a,bc
&b,bf
+c,11110
=a,a,bg
&bc,d
&a,ba
&b,bb
&c,ah
&d,ca
=be,1,c
&d,111101101
=a,a,d
&cb,10011
&cc,1011000
&cd,1
+cd,cd
&ce,1010
&cf,1
&a,cf
&b,ce
&c,1100000100
=a,a,cc
&cf,d
&a,cf
&b,cd
+c,11110
=a,a,cb
&c,1011100110
=d,1,c
&a,cd
&b,cf
&c,1101011110
&d,110101011
=a,a,d
&a,d
&b,1
+c,10111
=a,a,cb
&c,1011010011
=d,1,c
&c,1111010101
=d,0,c
&g,a
&a,101
&c,1110111001
=a,a,cc
&b,d
&a,g
&c,1101110101
=a,a,cb
%
```

References

[1] C.S. Calude, Elena Calude, M.J. Dinneen. A New Measure of the Difficulty of Problems, Journal for Multiple-Valued Logic and Soft Computing **12** (2006), 285-307

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

[3] C.S. Calude. Dyson Statements that are likely to be True but Unprovable (2008), see also http://www.cs.auckland.ac.nz/~cristian/fdyson.pdf

[4] C.S. Calude, Elena Calude. Evaluating the Complexity of Mathematical Problems.Part2, Draft, dated July 19,2009

[5] F. Dyson. in J. Brockman. What We Believe but Cannot Prove, Harper Perennial, New York, 2005, 82-83. See also http://www.edge.org/q2005/q05_9.html#dysonf

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

E-mail address: jhertel@h-star.com

Current address: H-Star,Inc 20801 Biscayne Blvd 4th Floor Aventura,FL 33180 USA