ASIAN JOURNAL OF MATHEMATICS AND APPLICATIONS
Volume 2013, Article ID ama0093, 10 pages
ISSN 2307-7743
http://scienceasia.asia

THE CONDITIONAL INDEPENDENCE OF THE RIEMANN HYPOTHESIS IN RELATION TO ZFC

CRISTIAN DUMITRESCU

Abstract. In this article I describe a proof of the fact that ZFC cannot decide whether a certain modified Turing machine, or computer (satisfying a certain condition related to the running time without halting, up to a certain threshold) will ever halt successfully in finite time. The consequences of this fact in relation to the Riemann Hypothesis are presented.

Introduction.

The Riemann Hypothesis has been an open problem for a long time. One of the reasons why this problem remained not solved for over 150 years is that, most likely, ZFC cannot prove that RH is false. In this article I present the main ideas in support of this view, that ZFC cannot prove that RH is false (and some metatheoretical consequences).

Section 1. Reformulations of the Riemann Hypothesis.

The following reformulations have been known for some time and the proofs of the statements in this section can be found in many references.

Lagarias' reformulation of RH [5]. The Riemann Hypothesis is true if and only if $\sigma(n) \leq H_n + \exp(H_n) \cdot \log(H_n)$, for all n (here $\sigma(n)$ is the sum of divisors function and H_n represent the harmonic sums).

Robin's reformulation of RH [7]. The Riemann Hypothesis is true if and only if there is an n_0 (and in fact $n_0 = 5040$) such that $\sigma(n)/n < e^{\gamma} \cdot \log(\log(n))$, for all $n > n_0$ (here $\sigma(n)$ is the sum of divisors function).

Littlewood's reformulation of RH [4]. The Riemann Hypothesis is equivalent to the statement that for every $\varepsilon > 0$, we have $M(x) = O(x^{(1/2+\varepsilon)})$, when $x \to \propto$ (here M(x) is the

2010 Mathematics Subject Classification: 11M99.
Key words and phrases: Riemann Hypothesis, Robin's reformulation, Lagarias' reformulation, Littlewood's reformulation, algorithmic complexity.

Mertens' function).

Section 2. Algorithmic Complexity, a very brief introduction.

The following very brief presentation of some fundamental notions of algorithmic information theory is based on references [1], [2], and [8].

Definition 1. A language L, finite or infinite, is called prefix - free if and only if no word in L is a prefix of another word in L.

Consider the binary alphabet $V = \{0, 1\}$. We consider the partial functions $f : V^* \times V^* \to V^*$, where V^* is the set of finite strings (words) of V, including the empty word. We also consider the projections $f_y(x) = f(x, y)$.

Definition 2. A computer is a partial recursive function $C : V^* \times V^* \to V^*$, such that, for all $v \in V^*$, the domain of C_v is prefix - free.

Concrete construction of a computer, as a modification of a Turing machine. We can think of u as the program and v as its input. The computation steps follow a finite table that completely determines the state changes starting with (u, v). The computer C has two tapes, a program tape and a work tape (each can only contain 0 or 1 in their squares). The program tape is a read - only tape, and the read - only tape head can only move to the right (or stay in the same position). Initially, the program u occupies the whole program tape, except for the leftmost blank square which is scanned by the reading head.

The work tape contains the input v, and the read - write head scans the leftmost square of the squares containing the symbols that represent the word v. The read - write head can move in both directions. The computer has finitely many internal states, among which there is an initial and a halt state.

There is a finite table that determines the state transitions of the computer, depending on the current state and the scanned symbols from the program and the work tape, which determine the next state, the symbols written on the work tape, the move of the read - only head, and the move of the read - write tape.

The computation of the computer is a success if C enters the halt state, when the read - only head is scanning the rightmost square of the program tape. If the computation is a failure, the halt configuration is not reached, then C(u, v) is not defined.

It can be shown that this concrete machine model computes exactly the partial recursive functions specified in the definition above.

Definition 3. A computer U is called universal if and only if, for every computer C, there is a simulation constant sim (C), such that, whenever C(u, v) is defined, there is a program u' such that U(u', v) = C(u, v), and $|u'| \leq |u| + \text{sim}(C)$.

Theorem 1 [8]. There is a universal computer.

Note. In the following, we will use the words *program* and *computer* interchangeably, we could look at a computer (its binary encoding), as a program for a universal computer that simulates it. In both cases we have in mind an *algorithm* that can perform a computation successfully or fail, or a *modified Turing machine*, as described in the previous section. It is also essential that in this article we only consider computers with no input. This is important in the presentation, in order to avoid any confusion.

We consider the lexicographic ordering on the set of words over $V = \{0, 1\}$.

Definition 4. Given $w \in V^*$, the canonical program w^* for w is defined by $w^* = \min \{u \in V^* \mid U(u, \lambda) = w\}$, where λ is the empty word. Thus, w^* is the first (in lexicographic order) program (and the shortest program) of the universal computer U producing w, when U started with the empty work tape (empty input).

Theorem 2 [8]. The function $f : V^* \to V^*$ defined by $f(w) = w^*$ is total. For any w, there is a w^* such that $U(w^*, \lambda) = w$, and $w^* \neq \lambda$.

Definition 5. The complexity of a word w with respect to a computer C equals
$H_C(w) = \min \{ |u| \mid u \in V^* \text{ and } C(u, \lambda) = w\}$. The min is undefined if the set involved is empty. We also define $H(w) = H_U(w)$, where U is a universal computer. H(w) is the length of the minimal program for U to compute w when started with an empty work tape.

We will not discuss further the conditional complexity of a word with respect to another word, the invariance theorem, or the properties related to these notions. Reference [8] is more than sufficient for clarification.

In the following, I assume that the reader is familiar with the fundamentals of Algorithmic Information Theory, prefix free codes and Chaitin complexity. In [9] it is stated the following theorem.

Theorem 3 [9]. Let T be a 1 - consistent theory and U a universal Chaitin computer. Then there is a positive constant C (depending only on the universal computer U) such that T can determine at most H(T) + C bits of Ω.

In the theorem above, H(T) represents the number of bits it takes to describe the arithmetical theorems of T. In other words, H(T) represents the algorithmic complexity of the program (modified Turing machine) that generates all the theorems of T systematically (from axioms and rules of inference). Even if, in general. H(T) in not computable, we can find upper bounds.

We consider now Lagarias' reformulation of RH above. We consider the modified Turing machine Z (that starts from an empty tape) that takes the positive integers n in sequence, calculates the sum of divisors $\sigma(n)$ and the harmonic sum H_n (for each n), and then checks whether the inequality $\sigma(n) \leq H_n + \exp(H_n) \cdot \log(H_n)$ is satisfied. If it is satisfied, then it follows the same procedure for n + 1, and so on. This modified Turing machine (Z) only stops if it finds an n such that $\sigma(n) > H_n + \exp(H_n) \cdot \log(H_n)$, otherwise it runs forever (it never stops).

We consider an enumeration of all computers P_1 , P_2 , P_3 ,…….P_n , ….., encoded as binary strings in lexicographic order. We emphasize that we only consider the computers that start from a blank tape (with no input). These computers can also be seen as programs for the universal computer U.

Define $\tau = \sum_{P_n \text{ halts}} 2^{-n}$. This number is an encoding of the answers to every instance of the halting problem in a single real number. The n-th bit of τ is 0 if the computer P_n runs forever (or the computation fails), and it has the value 1 if the computer P_n halts successfully in finite time (the computation id a success). As far as I know, Turing first considered this oracle [3].

The real number τ is not a random number. The complexity of $\tau(n)$, an initial segment of length n is around $\log_2(n)$ [6].

Theorem 4 [6]**.** The complexity of $\tau(n)$, an initial segment of length n is around $\log_2(n)$.

Proof. The first n bits of τ represent n instances of the halting problem, and to solve these, we only need to know how many of these n programs halt. We can then simulate each program in parallel until this many programs have halted. We then know that all the other programs under consideration will never stop. We conclude that n bits of τ are computable from $\log_2(n)$ bits of information. **QED.**

We write l(P) for the length of the binary encoding of the corresponding program P, and we write t(P) for the number of state transitions, (we will also call them time steps), before the program P halts. If P does not halt in the halting configuration in finite time, then t(P) is undefined.

Section 3. The modified Turing's oracle and the main result.

The lexicographic order relation of the listed programs will be represented by $<$. We will now define a new order relation $<^*$ on the same list of programs.

It is clear that for a successful program we have $t(P) \geq l(P)$ (from the definitions above). A failed program can fail because it does not stop in the halting configuration, or it runs forever.

We reorder **only** the successful programs P (for which $t(P) \geq l(P)$ and $t(P) < \infty$), in such a manner that $P_i <^* P_j$ (in the new order), iff $t(P_i) < t(P_j)$. In case $t(P_i) = t(P_j)$, then the programs will be listed in lexicographic order. Only the successful programs P for which $t(P) \geq l(P)$ and $t(P) < \infty$ might change their order rank in the new order $<^*$ (compared to the initial lexicographic order). As a subset, the subset of successful programs is invariant, as related to the original lexicographic order in the list of programs. Only within the subset of successful programs, the programs are permuted and reordered, according to the number of time steps before halting successfully. The number τ remains unchanged. As a subset, the subset of failed programs is invariant, as related to the original lexicographic order in the list of programs. Within the subset of failed programs, the programs maintain their initial lexicographic order. The number τ stays the same, with the new ordering of the list of programs.

The new order $<^*$ might not be computable from the lexicographic order $<$ (in the list of programs), but we assume the existence of an oracle O, that gives the list of all programs in the new order $<^*$. As can be seen, if the validity of the main theorem (in the following) can be established assuming the existence of the oracle O, then even more so, the main theorem is valid if we do not assume the existence of the oracle O.

If we know that a successful program P (assuming we know its binary encoding) halts in time t(P), and that t(P) ≥ l(P) and t(P) < ∞, then we can compute all the bits corresponding to all the programs before P (up to the program P) in the order relation <* defined above. We have the following theorem.

Theorem 5. (The τ - segment reconstruction theorem). We assume that we know that the program P is successful, and we know its binary encoding. Then we can reconstruct all the bits of τ, from the initial segment of bits of τ, up to the bit corresponding to the successful program P (when the programs are listed following the order relation <* defined above).

Proof. We perform the required permutation among the programs in the subset of all successful programs. If we know that P is a successful program, then we run P until it halts, and we find t(P), the number of time steps before P halts successfully. We then run all the programs up to length t(P) for t(P) time steps. Any program of length greater than t(P) will stop in more than t(P) time steps (if it is a successful program), so it comes after P in the defined new order <*. Among the programs of length less than t(P), we can then find all the successful programs, and their related number of time steps before halting. We have the order rank of our program P as related to the order relation <*. It is clear then, that all the programs before P (in the <* order) that did not halt in t(P) time steps (or failed early), are all failed programs. We then know the location of 0's and 1's in the initial segment of τ, up the bit corresponding to the successful program P considered above. **QED.**

We also have the following important theorem.

Theorem 6. We can only know that a finite number of programs are successful.

Proof. We can find the values for an infinity of bits of τ, but we can only know that a finite number of programs are successful. In other words, if we know that an infinity of bits of τ have value 1, and if we also know the corresponding programs that halt successfully (for which t(P) ≥ l(P) and t(P) < ∞), then we can reconstruct all the bits of the number τ (from theorem 5). This would be a contradiction, since the halting problem is unsolvable. **QED.**

We note that, in principle, we know that there are an infinity of successful programs (for example, programs that print a given word of any length), but we only consider computers with no input, so any complex dynamics must be embedded in its internal states. It follows that it is not a trivial matter to find the binary encoding of such programs (if we work within ZFC), in order to reconstruct a given segment of the list of programs, and the number τ.

We can then formulate the following main theorem.

Theorem 7 (The Main Theorem). If the modified Turing machine Z (defined above, as related to RH) satisfies the relation $t(Z) > 2^{H(ZFC) + C}$ (where the constant C can be determined, and depends on the universal computer U that we choose), then ZFC cannot determine whether the modified Turing machine (computer) Z eventually halts successfully in finite time. In other word, under these circumstances, ZFC might be able to prove that Z fails, but it will never prove that Z halts successfully.

Proof. We write P_{last} for the program that we can prove that is successful ($t(P_{last}) \geq l(P_{last})$ and $t(P_{last}) < \infty$), and has the highest order rank in the order relation <* defined above. We proved that this program exists. We want to find the conditions under which our modified Turing machine Z (our program Z) has an order rank (in the order <*) that is greater that the order rank of P_{last}.

We consider all the bits of τ, up to the bit corresponding to the program P_{last}, and we write N for its length. This initial segment of τ of length N has complexity around $\log_2(n)$. That means that no program much smaller $\log_2(n)$ can generate this initial segment of length N of τ. If a formal system like ZFC can generate an initial segment of τ of length N, then this N has a maximum value around $C' \cdot 2^{(H(ZFC))} = 2^{H(ZFC) + C}$ (where the constant C can be effectively determined).

As defined before, whether the program Z (related to the Lagarias' reformulation of RH) halts or runs forever is equivalent to the Riemann Hypothesis being false or true. It is easy to prove (a simple numerical experiment) that if Z ever halts, then Z is a program that satisfies the condition $t(Z) \geq l(Z)$ (we just let it run for $l(Z)$ time steps and see that it does not halt successfully).

If we can verify that Z also satisfies $t(Z) > 2^{H(ZFC) + C}$(as we assumed in the hypothesis of this theorem), then this means that the order rank of Z (for the order relation <* defined above) is greater than the length of the initial segment of τ for which ZFC can determine all the bits. We note that $rank(Z) > t(Z)$, we can see that from theorem 2. Basically, for any word of any length, we can think of a computer that starts with no input and prints that word, the whole action being embedded in its internal states. That means that for any length less than $t(Z)$, there is a successful computation with that many time steps (the argument can be further refined), so that any previous order rank is *occupied* by a successful program. Note that this does not mean that we can actually identify (find their binary encoding) all these printing program, we just know that they exist, and that they perform the corresponding successful computation in about the same number of time steps as the length of the word to be printed..

If we assume that $t(Z) > 2^{H(ZFC)+C}$, then we can conclude that ZFC cannot decide whether Z halts in finite time, because the order rank of Z is greater than the order rank of P_{last}, otherwise ZFC could reconstruct a larger initial segment of τ than proved possible. We could conclude then, that ZFC cannot prove that RH is false. **QED.**

Observations and conclusions. We used here Lagarias' reformulation of the Riemann Hypothesis, but we can use Robin's reformulation, or other reformulations [4], [5], [7]. The essential idea is to link the Riemann Hypothesis to a program (computer, algorithm) in such a manner that RH is false iff the program under consideration halts successfully in finite time (and RH is true iff the program under consideration fails by running forever).

We also note that verifying that the program Z does not halt in less that $2^{H(ZFC)+C}$ computation steps is not an easy task (this may be a task for a quantum computer, but impossible with current technology, still, we do not know what the future brings).

The conclusion is that ZFC cannot prove that a computer halts successfully if that computer takes a very long time (longer than the threshold above) to halt (even if, at least in principle, it might eventually halt in finite time). The very essence of how a formal system is defined must be modified, in order to deal with these problems, but this is a different challenge altogether.

Another conclusion is that, since **if** t(Z) is finite (even larger than the threshold above), then this finite computation would represent by itself a proof that RH is false (within ZFC), the only conclusion that we can reach is that the Riemann Hypothesis is true (as a metatheoretical argument). Refinements of this method can probably lower the value of the threshold value of the number of time steps. In problems of this type, a very large, but finite computation, would be sufficient, in order to settle the Riemann Hypothesis. We can estimate an upper bound for H(ZFC), and the constant C (by choosing a particular universal computer U).

We emphasize that the results in this article do not prove that ZFC cannot prove that RH is true, so a direct proof (not metatheoretical) that RH is true might still be possible within ZFC.

Many conjectures in number theory (including RH) can be linked to the problem of whether a certain computer performs a successful computation or fails ([1]). We let that particular computer run for $2^{H(ZFC) + C}$ time steps, and if the computation does not halt successfully up to that point, then we can conclude that the original conjecture in number theory cannot be disproved in ZFC. As a conclusion, we might as well take the statement of that particular conjecture as an axiom (or a true statement, based on metatheoretical arguments), since we are sure that the new system will be consistent, and continue to study new conjectures in the new system (with a higher threshold).

I will conclude with some philosophical remarks of a speculative nature. As an estimation, $2^{H(ZFC) + C}$ is of the order of 2^{10000} , but with a suitable choice of the universal modified Turing machine in the context of AIT, we can probably decrease this threshold to about 2^{1000}. In his work, Seth Lloyd estimated that the informational content (or computational power) of our Universe (our Hubble bubble) is around 2^{400} bits. Our Platonic view of number theory (and mathematics in general) must change, and allow an understanding of mathematics closer to physics. Following Landauer and P. Davis, mathematics is meaningful only if it is the product of real computational processes, rather than existing in a Platonic realm. There is then a self - consistency argument that must be incorporated in a larger program directed at unifying mathematics and physics. It is not true that no matter how much numerical evidence we have in favor of a conjecture, we might find some counterexample for larger values of the parameters. For certain problems (and there are many in number theory, including RH) there is a threshold of numerical evidence (as shown above, the value $2^{H(ZFC) + C}$) that is sufficient to guarantee that ZFC cannot prove otherwise (the negation of the conjecture), and we can add the mathematical statement under consideration as an axiom within an extended axiomatic system. The main challenge is to decrease this number (the threshold) to a smaller number, so that the threshold can be accessed with current computers. In one sentence, we can reformulate the conclusions that we reached as *"in mathematics, there is no proof of a statement, that is greater (as informational content, or computational power) than the computational power of the Universe in which we live"*.

REFERENCES

[1] C. H. Bennett, "On Random and Hard to Describe Numbers", in C. S. Calude, *"Randomness and Complexity, from Leibniz to Chaitin"*, World Scientific, Singapore, 2007.

[2] G. Chaitin, *"A Theory of program size formally identical to information theory"*, Journal of A.C.M. 22 (1975), 329 - 340.

[3] B. J. Copeland, D. Proudfoot *"Alan Turing's Forgotten Ideas in Computer Science"*, Scientific American, April 1999, 98 - 103.

[4] H. M. Edwards, *"Riemann's Zeta Function"*, Dover Publications, Inc., 2001.

[5] J. C. Lagarias, *"An elementary problem equivalent to the Riemann Hypothesis"*, American Math. Monthly,

109 (2002), 534 - 543.

[6] T. Ord, T. D. Kieu, *"On the Existence of a New Family of Diophantine Equations for Ω"*, in C. S. Calude, *"Randomness and Complexity, from Leibniz to Chaitin"*, World Scientific, Singapore, 2007.

[7] G. Robin, *"Sur L'Ordre Maximum de la Fonction Somme des Diviseurs",* Seminaire Delange-Pisot-Poitou, Theorie des nombres (1981 - 1982), Progress in Mathematics 38 (1983), 233 - 244.

[8] G. Rozenberg, A. Salomaa *"The Secret Number. An Exposition of Chaitin's Theory"*, in C. S. Calude, *"Randomness and Complexity, from Leibniz to Chaitin"*, World Scientific, Singapore, 2007.

[9] R. M. Solovay, *"A version of Ω for which ZFC can not predict a single bit"*, in C. S. Calude, G. Paun (eds.), *"Finite versus Infinite. Contributions to an Eternal Dilemma"*, Springer - Verlag, London, 2000, 323 - 334.

119 YOUNG ST., AP. 11, KITCHENER, ONTARIO N2H 4Z3, CANADA