**Cantor's Proof—**

**From a Computer-Science Perspective**

Warren Robinett and Henry Lieberman, May 3, 2022

**Abstract**

This paper is one component of a multimedia work, which consists of this paper, an interactive simulator, and two videos. We have approached Cantor's 1891 Uncountability Proof using a standard technique in Computer Science—namely, by making a *simulator* for Cantor's Diagonal-Method construction. The job of the simulator is to provide a tool for exploring the computation performed by the Diagonal Method, and also a means of visualizing this computation. The videos show how to use the simulator, and what can be learned from using it. The job of this paper is to set the stage, mathematically, for using the simulator to analyze the Diagonal-Method algorithm's behavior.

*Setting the stage* includes several topics. (1) We discuss which representations constitute a valid definition of a real number. (2) We describe a mathematical maneuver that we use to simplify the problem, and thereby reduce the problem from one involving a table of infinite width and height to a finite (1010) table; and at the same time, we transform the problem from one involving an arbitrary (but unspecified) enumeration to a concrete set of real numbers occupying the 1010 table. From this finite and concrete beginning, the simulator shows us, in great detail, the behavior of the Diagonal Method algorithm. When we focus the simulator on the bit that Cantor built his proof around (which we call the *critical bit*) we find that this bit is indeterminate—because when computing this particular bit, the Diagonal-Method algorithm fails to generate a digit value. *The cause of this failure is an attempt to read an uninitialized variable.* This discovery means that the supposed real number generated by the Diagonal Method is ill-defined; and so there is no contradiction; and the proof establishes nothing.

In overview, the failure in Cantor's Proof may be described as follows. The Diagonal Method construction in Cantor's Proof contains self-reference; and when the algorithm is expressed in a modern programming language, the self-reference leads to a run-time error when attempting to compute one particular digit in the generated real number. This digit is thus *indeterminate* in value, and so the containing real number is *ill-defined*. This failure in the Diagonal Method construction invalidates Cantor's 1891 Proof, but leaves open the possibility of other (non-diagonal) proofs of the theorem. The failure also calls in question other proofs based on the Diagonal Method, such as Gödel’s 1931 Incompleteness Proof and Turing's 1936 Halting-Problem proof.

**1. Introduction**

The currently-accepted foundations of mathematics are based on pre-computer ideas. This pre-computer period ended around 1940. We propose to audit Cantor's Uncountability Proof, using the new ideas introduced by emergence of the electronic computer, and the new understandings possessed by the field of Computer Science, all of which has occurred since World War 2.

Through the example of computer programs, we now have a better understanding of *what computation is*. We will therefore analyze Cantor's Diagonal Method construction *as an algorithm*. We will ask: Is this algorithm well-defined? Is it a total function?

There are tensions between mathematics and Computer Science. This is OK: and perhaps even valuable. A debate between the two points of view—sometimes called a dialectic—could possibly lead to resolution and mutual accommodation.

*The above justification is a sufficient reason to critically examine Cantor's Proof*. And furthermore, permission to do so is not required. If we find no errors—no harm is done. If we do find errors, then we learned something we didn't know before. If you think back to Euler's time, the critique of the initial 1758 version of Euler's Theorem (the equation relating the number of vertices, edges, and faces of a polyhedron) yielded an important discovery (a link between geometry and topology) [REF Lakatos 1976]. There are good reasons to audit, and it has no downside.

**1.1. Representing Real Numbers**

Reasoning about the set of ALL real numbers requires a *definition* of what a real number is. Without a clear definition, we won't know if we got them all; or what criterion to use to exclude entities we wish to bar from membership in the set.

Does it work to define a real number as an infinite sequence of digits? To a computer scientist, the answer is *NO*—because a truly infinite sequence of digits *NEVER ENDS*—there is no way to write them all down, or store them in a computer memory. If you stop after giving the first 10100 or 101000 or 101,000,000 digits, *you have not defined an infinite sequence*—sorry, but you have merely defined a finite sequence.

A better idea is to give a *procedure* for generating these digits. This definition of a real number allows us to create a finite representation (the procedure) of an infinite sequence of digits. In modern terms, the procedure is a *program*. Finding a way to represent an infinite structure with finite means is actually a brilliant intellectual achievement. This representation of real numbers was used by Turing in his 1936 paper [REF], which introduced the idea of the Turing Machine.

It doesn't matter whether we use base-10 or base-2 (as Cantor did) to represent real numbers; the two representations are equivalent [REF Martin Davis]. We will use binary numbers here—because arithmetic is implemented in base-2 on modern computers.

**1.2. The Word "All"**

The word *ALL* denotes an important and powerful concept in mathematics. But along with the power come complications and danger. When you say ALL, you have to get it right in every single case. One failure—even alongside an infinite number of successful cases—can kill your theorem.

In particular, when you define a real number, the definition had better produce a well-defined digit—for every one of the positions in the binary number. If even *one digit* is undefined—then it is not a well-defined real number. In such a case, we say that the digit is *indeterminate*, and the real number as a whole is *ill-defined*, or *undefined*.

The word ALL also brings along obligations. Suppose we had a procedure that, starting from a list of real numbers, could create a new real number that was not in the original list. If this new construction was truly a well-defined real number, then *we would have to say* that it was contained in the *set* of ALL real numbers: . And if we had a *list* of ALL real numbers, the new real would have to be somewhere on that list. The meaning of the word ALL requires this.

Cantor's Proof is a proof-by-contradiction, in which the initial assumption is that there exists an enumeration of the real numbers in [0,1]. This enumeration, by definition, must include ALL real numbers in [0,1]. Cantor then used the Diagonal Method to construct a new real number in [0,1], which we will call Z, and he claimed that Z is different from all the real numbers in the assumed enumeration. If Z is truly a well-defined real number, then it has to be somewhere in the enumeration, at some index *k*. This is required by the meaning of the word ALL.

Once we know that the constructed real Z is located in the Cantor table at some row *k*, we can then ask a question: *What happens at the table entry where the diagonal of the table crosses row k?*

The program that generates the new real number Z—that is, the program which generates the digits in row *k* of the table— is relatively simple. We can write it down, in the programming language C, as:

T[k][i] = ! T[i][i] ,

where T[k][i]denotes the -th digit of row ; and T[i][i] denotes the -th digit of the table's diagonal. The operator ! is C's symbol for the NOT operation. T stands for the whole Cantor's Table, a two-dimensional array of infinite width and height.

**1.3. Simplifying the Problem**

Before we go any further, we are going to perform a maneuver to simplify the problem. We are going to permute the assumed enumeration given to us by Cantor. A permuted enumeration is still an enumeration—so this is a legal move, mathematically speaking. We are going to choose a permutation that moves some specific real numbers into the first ten slots of the enumeration—rows 0 to 9.

We move the diagonalization construction, defined by program T[k,i] = ! T[i,i], into row 3. This makes .

Into rows 0 to 2, we put the real numbers and . In the remaining rows (4-9), we put the real numbers and .

Carrying out this permutation gives us some *definite real numbers* in the first ten row of the enumeration. It also does one other very important thing. It allows us to reduce the problem from one involving an infinite dataset (the entire Cantor Table) to a small subset of the Cantor Table (in its upper left corner).

The reasoning is this. Cantor's Proof is claimed to apply to ANY enumeration of the real numbers; so it applies to this enumeration that we created using a permutation. If we can show that there is a failure in the subset of this enumeration, then we have shown a failure in the general proof of Cantor's Theorem.

This maneuver also gets around one of the obstacles that has frustrated critics of Cantor's Proof for many years—the lack of a concrete enumeration of the reals upon which to test the claimed construction. For the first ten rows of this (permuted) table, it gives a concrete beginning to an enumeration of the real numbers. This subset will be enough for us to show that something goes wrong in Cantor's construction.

**1.4. The Critical Bit**

Cantor's Proof is entirely based on one binary digit in the table; in our permuted table, this digit is located at table location T[3][3]. We call it the *critical bit.* Or for short—the *critbit*. We also give this bit a second name—bit .

We agree with Cantor that this table location T[3,3] is of critical importance to Cantor's Proof; but whereas Cantor claimed that this bit led to a contradiction, we say that Cantor's analysis was wrong—and that the value of this bit is actually *indeterminate*, which implies that the real number (supposedly) defined by row 3 is ill-defined. An ill-defined real number is *not a number*, and it cannot be proven to have any properties at all. So there is no contradiction, and the proof establishes nothing.

**1.5. Trying to Compute the Critbit's Value**

According to our analysis, if you look carefully at Cantor's Diagonal Method construction, you can see that the critbit is actually defined in terms of itself; as

b = ! b

Because each table location is a binary digit, the only possible values are 0 and 1. But this definition doesn't successfully define the value as either 0 or 1.

Watch Video #1 to see a depiction of the failure mechanism, and to see how this failure can be demonstrated using the simulator.

We have to conclude that:

* + The value of the critical bit *b* is *indeterminate.*
  + The value of the real number containing *b* is *ill-defined*.

But some may doubt that our analysis is correct. To settle this sort of question, we have created a *simulator*—the Diagonalization Simulator, which we call *DiagSim*. It can be accessed at [diagsim.com](http://diagsim.com/). Its purpose is to inspect the details of the computation of values in Cantor's Diagonal Method algorithm.

**2. The *DiagSim* Simulator**

*DiagSim* is an interactive simulator for Cantor's Diagonal Method algorithm (Fig. 1); it also provides a visualization of the computation. Using the simulator, we can see the computation of the Cantor Table, step-by-step and digit-by-digit.

For the reasons discussed in Sec. 1, we are simulating only a 1010 subtable of the Cantor Table. This subtable consists of the first 10 binary digits of the first 10 real numbers (rows) of the Cantor Table. The best way to understand what the simulator can do, and how to use it, is to watch the videos on the website—starting with Video #1 (<http://unmander.com/diagSim/video/diagsim_video1.mp4>). We will also give here a brief overview of the simulator.

*Main Table Area.* Following Cantor, all the real numbers in the table fall within the unit interval [0,1]. Each row shows the first 10 binary digits of a real number, and the generating program for each number is to the left of the row. The programs are written in the C programming language. To the right of the main table, the generated numbers are shown a second time—but in this area, the numbers may be displayed in one of several formats: binary fraction, decimal fraction, or rational fraction. (These values are approximations, based on the first 10 binary digits.) The format used is controlled by a GUI widget labelled Output format.

*Whole Table Execution Timeline*. The 1010 table consists of 100 cells, each holding a single binary digit—which shows the calculated value of that digit, when the computation of the table is complete. Before computation has begun, the table shows the initial value of each cell: either 0, 1, or — (with "—" meaning *uninitialized*). The app is a *visual simulator of the Diagonal Method computation*, and thus shows a visualization of the computation as it proceeds. This includes showing the table when the computation is partially done, and animating the computation as it progresses.

Graphical user interface

Description automatically generated

Fig. 1. The *DiagSim* simulator. The simulator is available at [diagsim.com](http://diagsim.com/).



The computation is thus portrayed as a "movie," which explains why it has VCR-like controls to start, stop, and back up the animation. The ability to back up the animation implies that the simulator is a reversible interpreter [REF, Henry 1987 paper]. The computation can also be advanced, or run backwards, using the slider control near the VCR buttons. A third way to move the computation forward or backward is to click on a cell in the table, which moves the computation state to the point in the overall computation when that cell was being computed. The cell thus selected is called the *active cell*.

Text

Description automatically generated

*Cell-Value Computation Timeline*. The computation of the digit in the active cell may be inspected by selecting trace: single-step in the Cell Value Computation area. Doing this makes visible a second set of VCR buttons and adjacent slider. Using these controls, the computation of the active cell's value may be single-stepped, or backed up, using the yellow VCR buttons and yellow slider. The cell-value computation, as it progresses, is shown in the adjacent panels—labelled stack trace, stack, token-queue, active program, and status.

*Main GUI Area, Column 1*. The cells of the table may be executed in different orders. Since some table values depend on other table values, *the order of execution can make a difference in the final values of some cells*. Ten different execution orders are available, and the one used is controlled by the Table execution order widget. The widget Initial cell value controls which value is used for the initial value of the cells, before execution begins. The Speed widget controls how fast the animation proceeds.

*Main GUI Area, Column 2.* The last column of the GUI area provides various visualization options. The colors red, orange, and yellow are used as color codes to indicate three different types of errors. The Error visualization widget turns on a visualization of dependencies between table elements. The Table access visualization widget turns on a data-flow visualization showing how data values move into and out of the table.

Again, the videos (available at the diagsim.com website) show how to use the simulator.

**3. The Failure of the Diagonal Method Construction**

What we learn from the simulator is that *one of the bits in the table is undefined*—because the definition b = ! b does not define any value at all. This is the bit at location . To define a real number, you have to specify the value of *EVERY* digit. But for the real number supposedly defined in row 3—by the Diagonal Method algorithm—one of its bits is *indeterminate*. That bit is the critbit , also known as . You can use the simulator to single-step through the computation of the value of cell —and see precisely what occurs. What happens is that a run-time error occurs: an attempt to read an uninitialized variable. (Go try it on the simulator, if you haven't already done that.)

This means that, in row 3, *a well-defined real number was not successfully constructed*. So there is no contradiction. With no contradiction, a proof-by-contradiction does not work—it proves nothing.

Our most important result is that we have refuted, not just Cantor's 1891 proof—but more broadly, the *Diagonal Method proof technique*. This proof technique was widely used after Cantor introduced it in 1891. In particular, variants of it were used by Gödel in his 1931 Incompleteness Proof, and by Turing in his 1936 Halting-Problem Theorem. A refutation of the Diagonal Method proof technique would appear to demand an audit of all later proofs that employed the Diagonal Method.

For Cantor's 1891 proof itself, unless there is some other valid proof of Cantor's Theorem—it must be demoted to being a conjecture of unknown validity. But Cantor actually published two earlier proofs of the uncountability of the reals—in 1872 and 1884. We believe that we can also refute these two proofs, but this is outside this paper's scope.

But even if we had refuted all three attempted proofs by Cantor of the Uncountability Theorem, the possibility would remain that the theorem is true. There are other proofs of Cantor's Uncountability Theorem that use other methods than what Cantor used in 1872, 1884, and 1891. This needs to be looked into—but it is outside the scope of this paper. Refuting the Diagonal Method proof technique is the topic of this paper.

**3.1 Where Was The Error In Cantor's Proof?**

The fatal error occurs when the sequence of bits generated by the Diagonal Method algorithm is *inserted into the table*. This creates the (ill-defined) critical bit. If you copied these bits to a memory location EXTERNAL to the table, there is no problem. But when you say that this "new real number" must be somewhere in the table—because the table is a list of ALL real numbers in [0,1]—then we have a problem. There is one entry in the table where the diagonal of the table intersects with row *k*—this is the critical bit *b*.

The construction procedure is not complicated. The Diagonal Method algorithm tells you to grab that bit, inverted it, and stick it back into the same location—location T[3,3]—also known as the critical bit *b*. This operation can be written in symbols as

b = ! b , where is the NOT operation in C.

You don't have to be a rocket scientist to see that this doesn't define any value at all.

And *why* did we insert it into the table? *We had to*—because of the claim that the table contains *ALL* real numbers in [0,1]. If the new number generated by the Diagonal Method Algorithm is a well-defined real number in [0,1], it has to be somewhere on that infinite list—in some row *k*. It is the *COMBINATION* of using the Diagonal Method Algorithm *AND* locating it in a row of the table that causes the failure.

Did Cantor claim that the table contained all real numbers in [0,1]? Yes—in effect, he did, when he said, in the first sentence of his paper:

...there is an infinite set of elements which cannot be put into a one-to-one correspondence with the set of all finite whole numbers... [REF James R. Meyer translation]

The infinite set he refers to is the real numbers in [0,1]; and the enumeration he is setting up—the "one-to-one correspondence"—is, by definition, a relation between the *TOTALITY* of the two sets involved.

If the new real number he claimed to generate was truly a well-defined one—it had to be somewhere in the list of real numbers, because an enumeration contains ALL real numbers in the interval. And then it is perfectly reasonable to ask what is the value of the *k*-th bit of row *k*.

To put it into plain Anglo-Saxon—Cantor was already screwed—when that tricky little word "all" appeared in the reasoning. That is what caused the critbit's definition to refer to itself. That is what caused the self-reference to occur. What we see is that, whether he knew it or not, Cantor was playing with fire—that is, self-reference—and now, after a rather long delay, his theorem is going up in smoke.

The fatal mistake was allowing self-reference to creep into his definition—the same mistake that forced Russell to re-jigger the definition of the set in 1901.

**3.2 Contemporaneous Objections to Cantor's Proof**

Using b = ! b as a definition of a digit is such as elementary mistake that the professors of mathematics around the world—at Berkeley and MIT, at Oxford and Cambridge, in Paris and Berlin, in Moscow and Tokyo and Beijing—*will not be able to believe* that this mistake could have been missed by all the best mathematicians of the last 130 years.

*But it is a myth that no one objected to Cantor's results.*

* Kronecker, Cantor's teacher, said: *I don't know what predominates in Cantor's theory – philosophy or theology, but I am sure that there is no mathematics there*.
* Poincare said: *This will later be regarded as a disease from which one has recovered.*
* But Hilbert, the dominant mathematician of the day, said: *No one will drive us from the paradise which Cantor created for us.*
* And then Wittgenstein, the dominant philosopher of the day, replied: *If one person can see it as a paradise of mathematicians, why should not another see it as a joke?* Wittgenstein also described the diagonal argument as *hocus pocus*.

It is not really a very good defense of Cantor's Proof, to say that the mistake is so simple that if it was really a mistake, earlier mathematicians would have previously detected it. In fact, this is not a defense at all.

Our claimed bug in Cantor's Proof should be evaluated on its own merits. If you are able to take off the blinders—the ideas that were pounded into your head by your teachers—it is pretty obvious that b = ! b does not define any value at all.

It is asking for a Boolean value which is the opposite of itself. But no Boolean value with that property can exist!

**3.3. Hi There, I Have Some Bad News**

Standard procedure in the world of mathematics for dealing with someone who brings an attack on Cantor's Proof to a mathematics professor is to *SHOOT THE MESSENGER*. By *shoot the messenger*, we mean that the messenger is ridiculed, and met with an absolute refusal to listen to whatever arguments he (or she) may be advancing.

If you don't believe this, you should read the 1998 paper by English logician Wilfrid Hodges. The paper is called "An Editor Recalls Some Hopeless Papers," in which Hodges discusses the two dozen or so papers attempting to refute Cantor's Proof, that he received while editor of the *Bulletin of Symbolic Logic* during the previous 20 years.

This paper is interesting for several reasons. First, Hodges took the trouble to write it—when it wasn't likely to do him any good professionally, and might come back to haunt him. (Like, for example, now.) Second, he was *curious* about why so many people objected to Cantor's Proof. The paper begins:

I dedicate this essay to the two-dozen-odd people whose refutations of Cantor’s diagonal argument ... have come to me either as referee or as editor in the last twenty years or so. Sadly these submissions were all quite unpublishable; I sent them back with what I hope were helpful comments. A few years ago it occurred to me to wonder why so many people devote so much energy to refuting this harmless little argument—what had it done to make them angry with it? So I started to keep notes of these papers, in the hope that some pattern would emerge.

In spite of his evident skepticism, Hodges did read their papers—and even gave them guidance on how to properly refute a theorem:

It was surprising how many of our authors failed to realise that to attack an argument, you must find something wrong in it. Several authors believed that you can avoid a proof by simply doing something else.

Hodges should be commended for his curiosity and his sense of fairness—for it seems that he did evaluate the papers fairly, according to his own world-view and training. But the phrase *hopeless papers* in his paper's title reveals his overall attitude—complete certainty. His last sentence was:

Third, there is nothing wrong with Cantor’s argument.

We think that Hodges's curiosity was on target when he asked

why so many people devote so much energy to refuting this harmless little argument.

*Our answer* is that the *intuitive rejection* of the Cantor's Proof by so many people is a clue that something unusual is going on inside Cantor's Proof.

And we have followed Hodges's advice—we have attacked the arguments in the proof. We have used a relatively new style of attack, using computer-science ideas to micro-analyze Cantor's Proof, at the finest level of detail. And we have thereby found a glaring failure.

**3.4. Which Assumption Gets the Blame?**

If you insist on seeing Cantor's Proof as a proof-by-contradiction that DOES contain an actual contradiction, the faulty assumption is *NOT* the one that is usually singled out—that the real numbers are countable.

The faulty assumption is that *a Boolean variable can have a value with the property* b = ! b.

This is the condition that Cantor imposed. But it is an impossible condition. This can be proven by observing that

is a tautology.

In a proof-by-contradiction, you don't get to negate just any hypothesis you want—you have to use the nearest mistake in the up-stream chain of logic. In this case, the nearest one is not the assumption about the countability of the reals—it is the mistake introduced by the Diagonal Method Algorithm—that a Boolean variable can have a value that satisfies the condition

b = ! b

But this is impossible. No Boolean value can do that. This is the faulty assumption in Cantor's Proof that should be given the blame for the claimed contradiction.

**4. Conclusions**

Computer Science gives us a new way to understand why Cantor's Proof is a failed proof.

As we have seen, Cantor's Diagonal Method construction is NOT well-defined, when looked at with modern eyes—that is, when analyzed as an algorithm, from a Computer-Science point of view.

The concepts necessary for this analysis were not available prior to 1940. They were not available until the electronic computer (and Computer Science) came along—in the decades after World War 2.

Today (in 2022), with 80 years of experience with computers to enlighten us, *we have a better understanding of what constitutes a well-defined computation*. And we saw that the algorithm used by Cantor was not actually well-defined—for one bit in the Cantor Table. But his proof was based entirely on that one bit—the critical bit—and so the proof falls apart.

This paper is one component of a multi-media work—consisting of this paper, an interactive simulator, and a video. This composite work is a new kind of proof—using not just words and symbols, but also dynamic visualization and interactive simulation—and it thus demonstrates a new way of investigating what is true, and what is not.

**5. References**

TBD

[1] Video#1. (The failure mechanism in Cantor's Proof—an uninitialized variable is attempted to be read.)

[2] Video#2. (How the DiagSim simulator works; using it to show the failure at the critbit.)

[3] The diagonalization simulator DiagSim, an interactive app, accessible at diagsim.com.

----------------------------

References for quotations.

James R. Meyer, translation of Cantor's 1891 paper into English.

FLESH OUT THE REFERENCES.

