A Stronger Foundation for Computer Science and P=NP

Mark Inman, Ph.D.
mark.inman@egs.edu
August 14, 2017

Abstract

This article constructs a Turing Machine which can solve for $\beta'$ which is RE-complete. Such a machine is only possible if there is something wrong with the foundations of computer science and mathematics. We therefore check our work by looking very closely at Cantor’s diagonalization and construct a novel formal language as an Abelian group which allows us, through equivalence relations, to provide a non-trivial counterexample to Cantor’s argument. As if that wasn’t enough, we then discover that the impredicative nature of Gödel’s diagonalization lemma leads to logical tautology, invalidating any meaning behind the method, leaving no doubt that diagonalization is flawed. Our discovery in regards to these foundational arguments opens the door to solving the P vs NP problem.

1 Turing’s Proof on the Entscheidungsproblem has a Fatal Flaw

1.1 Overview

1.1.1 Context

Turing’s monumental 1936 paper set the mechanistic description of computation which directly lead to the development of programmable computers. His motivation was the logic problem known as the Entscheidungsproblem, which asks if there exists an algorithm which can determine if any input of first order logic is valid or invalid. After defining automated computing, he described a program called an $\mathcal{H}$ Machine which can validate or invalidate its inputs based on reading a description number of that program. However, he did not provide an actual construction of such a machine, he only presumed its possibility to exist. His proof then depended upon this $\mathcal{H}$ Machine not being able to validate itself. He gives a detailed description as to why it can not validate itself.

However, a close reading of his paper shows an added assumption by Turing when he constructs his $\mathcal{H}$ Machine. While this assumption does not effect the construction or effectiveness of a Universal Turing Machine, it does have an effect on the overall results.
regarding the Halting problem and its sister problem, the Entscheidungsproblem, as well as any related results having to do with computational complexity.

The significance of this discovery of a fatal assumption in Turing’s work, should not be taken lightly. It effects more than just the proof on the Entscheidungsproblem, which should remain open after this discovery, but it also effects existing, accepted results on computability and the very foundations of key concepts in computer science such as complexity. This means we must write new textbooks.

In this article, we point out Turing’s assumption and then construct a description of a machine which does exactly what the assumption assumes is impossible. Because we have found an erroneous assumption, we must disregard Turing’s final results and any results which are derived from his method, including the Space Hierarchy Theorem which derives from Turing’s method using the same kind of assumption.

The construction of such a machine may have application in fault-tolerance of run-time self-correcting code validation in artificial intelligence implementations. It may also lead to better understanding of complexity relationships between complexity classes, as several results will have to be modified in response.

Furthermore, in a following section, we review methods which are differently constructed than Turing’s method, but are considered to be reducible to Turing’s method, such as Cantor’s Diagonalization Argument (CDA), and also Gödel’s Diagonalization Lemma (GDL). By addressing all three methods for proof in one article, we have effectively deconstructed the current foundations in computer science when it comes to the limits of computability. If we only deconstructed one method, then there might be reason to ignore our results on the foundation that it does not agree with the other two methods. However, we can and will address all three, and invalidate them all by providing non-trivial counter examples to the false assumptions in the methods.

1.1.2 Preliminary Considerations

The terms Circular Machine and Circle-free Machine are suitable for our description and we will use Turing’s own definition of a computing machine. It is convenient to note here that a Circular Machine is deemed by Turing to be unsatisfactory due to forever looping, redundantly over a repeating pattern. Also, a Circle-free Machine is deemed by Turing to be satisfactory because of its ability to continue deciding indefinitely, without entering an infinite loop. We have chosen to keep Turing’s original terminology for the sake of clarity when comparing the work of this article to that of Turing’s original paper. Also, we choose his terminology because Turing’s description of the Halting problem is completely mechanical, while many modern descriptions rely on an oracle, Cantor’s Diagonalization or logic similar to Gödel’s Diagonalization Lemma. Turing’s description is independent from these reductions in significant ways as a mechanical process. This helps the reader directly compare this article with the original proof without intermediary interpretations or simplifications. 

A Standard Description or S.D. is the rule set for any given Turing Machine \( \mathcal{M} \) in a standard form. By creating a standard, the rule sets themselves can be used to create a Description Number or D.N. which itself may be readable by a Universal Turing Machine, \( \mathcal{U} \), as an instruction set.

From Turing’s paper:
“Let $D$ be the Turing Machine which when supplied with the Standard Description (S.D.) of any computing machine $M$ will test this S.D. and if $M$ is circular will mark the S.D. with the symbol ‘$u$’ and if it is circle free, will mark it with ‘$s$’ for ‘unsatisfactory’ and ‘satisfactory’ respectively. By combining machines $D$ and $U$, we could construct a machine $H$ to compute the sequence of $\beta$.”

1.1.3 Turing’s Claim

Turing claims that while $H$ is circle free by construction, when $H$ is given the description number for $H$, it becomes circular. In the eighth section of Turing’s paper on the Entscheidungsproblem, Turing claims that $\beta$ can not be determined because of the following reason:

“The instructions for calculating the R(K)-th [figure] would amount to ‘calculate the first R(K)-th figures computed by $H$ and write down the R(K)-th’. This R(K)-th would never be found. I.e. $H$ is circular...”

This is because, since $H$ relies on certain subroutines to make its determination, when it reaches and tries to evaluate K, it must call itself, which provides instructions on reading inputs from 1 to K-1 in order to call the R(K)-th figure, but it can never get there, because it keeps repeating its own instruction loop.

1.1.4 Turing’s False Assumption

Turing assumed that there is no program that exists which can recognize itself arbitrarily and move to a circle free state upon this recognition. He assumed that any program would have to be programmed in such a way that when it reads itself, and calls its own instructions, it must be circular when trying to determine if it is circular or not, as described in the previous subsection. However, this is not necessarily the case and if we can provide an example of a program which does recognize itself arbitrarily so that it can switch to a circle-free state, then we’ve discovered a means to write $H$ machine in such a way that it may solve for $\beta$.

The question is, then, is there a Turing Machine which can recognize itself arbitrarily when it reaches its own Description Number (D.N.), such that some $H$ machine configuration prints $\beta$?

We will, in the next subsection, construct a Supermachine that can recognize itself as its own input which is then instructed to change to a circle free state upon this recognition. Because such a construction exists, and because it is arbitrary for any construction of this class of Turing Machines, we may solve for $\beta$ non-trivially.

---

1 by recognizing itself arbitrarily, we mean that it can recognize its own program or description number even if K is not fixed. Also, there may not exist an initializer that feeds a fixed K to be recognized by a single read instruction that skips K and just rubber stamps approval. Such “rubber stamping” is considered a trivial case and is not of concern to this article.
1.2 The Existence of Self-validating Computers

1.2.1 Supermachine

Let us consider that $H'$ is a controller machine with a D.N. of $K'$. It controls two different $H$ machines: $H_0$ and $H_1$. $H_0$ and $H_1$ each have the ability to determine “$u$” or “$s$” on a D.N. input, except $H_0$ tests as Turing describes, from D.N. 1 counting upwards (Each D.N. is a natural number) and $H_1$ tests from a certain two's complement of whatever number is being tested by $H_0$ as a simultaneous parallel input, such that its subsequent D.N. is one less than the previously tested D.N. Let us represent each D.N. by some integer $i$. $H_0$ and $H_1$ have a unique D.N. of $K_0$ or $K_1$ respectively.

Upon input of $i_0$ to be read by $H_0$, let $H'$ store the value pair $(i_0, z)$ until $i_0$ is determined to be satisfactory or unsatisfactory. When the output is determined, let $H'$ replace the $(i_0, z)$ with the respective $(i_0, s)$ or $(i_0, u)$ in the data store, such that there is no longer a data store of $(i_0, z)$. Let the same process occur for $i_1$, such that $H'$ also initially stores each D.N. input with $(i_1, z)$ and $H_1$ reads $i_1$ to determine satisfactory or unsatisfactory, subsequently replacing the initial value pair with the respective value pair $(i_1, s)$ or $(i_1, u)$ depending on the output of $H_1$. A redundancy occurs when some $i_0 = i_1$.

Let $H'$ have the ability to compare value pairs such that the machine may recognize a redundancy when it occurs, and may also recognize when a value pair contains a $z$ value on the condition of such a redundancy. Let’s call this a $z$-check ability.

Let $H_s$ be the supermachine that is the configuration of all three $H$ Machines as described above and let $K_s$ be the D.N. for the supermachine.

Initialize the identifier strings such that $K_1 < K_0$.

Let the number of bits in $K_0 = n$. Let the twos complement of the first D.N. input to $H_0$, which is 1, be determined by $n$ such that it satisfies the equation $c = 2^n - 1$.

*Lemma.* $H_s$ proceeds circle free, until it reads $K_s$. If $c - K_0 > K_1$, then re-initialize the D.N. in either $K_0$ or $K_1$ such that $c - K_0 < K_1$. This guarantees that $H_0$ will read $K_1$ before $H_1$ reads $K_1$ and also guarantees $H_1$ will read $K_0$ before $H_0$ reads $K_0$. Let the controller $H'$ contain a memory command which stores the decision value pairs of $H_0$ and $H_1$. The controller may routinely check for a redundancy on the next input.

Now consider when $H_0$ reads $K_1$, and $K_1$ calls the D.N. for $H_0$. $H_0$ will call $H_1$, which will call $H'$, which will result in a $z$-check, recognizing that the value pair $(K_0, z)$ is already stored in memory, and therefore, since $K_0 < c$, we know that $K_0$ is the description number for itself, which is impossible to call by construction without calling $H_1$ first, which means it must be checking the description number for a machine which calls itself, namely $H_1$, which allows us to correctly store the value pair $(K_1, s)$. This same reasoning can be applied for when $H_1$ reads $K_0$, correctly storing the value pair $(K_0, s)$.

---

2This can be determined through a unique identifier string, which does not effect the machine’s function or performance, but differentiates the two machines from each other giving them each a unique D.N.

3one may re-initialize, if necessary, the D.N. by adding irrelevant description information into some S.D. yielding a different D.N. provided such information does not affect the integrity of the original S.D.
If however, the machine has determined a redundancy occurred on a value pair where the value is either \((i, s)\) or \((i, u)\) (i.e., a negative evaluation on the z-check, but the redundancy check is positive), then we have already evaluated this D.N. from the other \(H\) machine at the top level, and we no longer have to continue within the range 1 to \(c\), since they will all have been decided. The supermachine, at this point proceeds to utilize machine \(H_0\) and proceeds from D.N. input value \(c + 1\), and continues through the rest of all Description Numbers, \(c + 2\), \(c + 3\), etc... at least until it reaches its own D.N., \(K_s\), for no other D.N. should be problematic in determining the output decision. Thus, \(H_s\) proceeds circle free, at least until it reaches \(K_s\) which is easily constructed to be larger than \(c\). □

1.2.2 \(\beta'\) is Decidable

Proof. \(\beta'\) is Decidable. At the point \(K'\) is received as an input, it is determined satisfactory by either \(H_0\) or \(H_1\). Neither \(H_0\) nor \(H_1\) are called during this phase of the process.

By lemma, \(K_0\) is decided by \(H_1\), \(K_1\) is decided by \(H_0\) and \(H_s\) continues indefinitely until we reach \(K_s\), which describes \(H_s\). \(K_s\) is read by \(H'\) and as before, its Description Number is stored along with its temporary pair value of z until \(H_0\) or \(H_1\) returns a value for \(\beta'\) at that location. \(K_s\) is sent to be verified by \(H_0\), which when \(H'\) calls \(K_s\) for a second time, under the given recursive property of \(K_s\) which will eventually call itself, the z-check for value pair \((K_s, z)\) is recognized as both redundant and with a z value, stored by \(H'\) in the data store, but because the associated value is z, the z-check ability tells us this process has already occurred, sends \(K_s\) to \(H_1\), which self-verifies repeated z-check values. By construction, the only value \(K_i\) which can provide this multiple z-check values where \(K_i > c\) is \(K_s\), so \(H_s\) now self-verifies the input \(K_s\) as its own D.N., provides a value of “s” for satisfactory, and changes state to evaluate \(K_s + 1\) to continue indefinitely as a Circle-free Turing Machine.

Therefore, given some Universal Turing Machine which can emulate the \(H_s\) Machine, \(\beta'\) is decidable over the set of all Description Numbers.

1.3 Consequences

Solving for \(\beta'\) is RE-complete. If Turing were correct, it would be impossible to solve for this output. Since we have solved for \(\beta'\), we must consider the possibility that there is a problem with our configuration which makes it impossible, or that there is something fundamentally incorrect with our current foundations of mathematics.

For this reason, we continue in the next section to examine Cantor’s Diagonal Argument, of which Turing’s method is equivalent, as they can reduce the same results to each other. If there is nothing fundamentally wrong with the foundations of mathematics and computer science, we should expect that any counter examples to CDA will not exist, or will be trivial. However, if we can find counterexamples for arbitrary
Figure 1: A supermachine configuration appears to exist
solutions, then such solutions will not be trivial and will reinforce the findings of this first section on the Entscheidungsproblem.

## 2 A Non-Trivial Counterexample to Cantor’s Diagonalization Argument

### 2.1 Overview

#### 2.1.1 Method and Foundations

Conventional thought on CDA is that we would need an entirely new axiom schema to find a counterexample to the method \[2\]. However, no new axioms are needed in this article to find a counterexample to CDA. We present in this section, a new grammar to generate a formal language for representation of an \(\omega\)-regular language through accepted foundations in set theory and the principles of formal languages.

For logical consistency, in order to examine CDA, we can not assume outright that CDA is a method which produces theorems, nor can we rely on any theorems derived from CDA or those theorems that reduce to it. This includes Turing’s proof on the Entscheidungsproblem, as discussed in a pervious section, in our exploration of it, as well as the downward Lowenheim-Skolem Theorem, or complexity results which rely on diagonalization such as the Space Hierarchy Theorem, et al. We must also exclude consequences of Cantor’s First Proof of Uncountability (CFPU) because of its similarity to the method of CDA. This requires that we assume the cardinality of the Real numbers as it relates to the Natural numbers is unknown, since CDA and CFPU are the very foundation that prove the Reals are strictly larger than the Natural numbers. This is for logical consistency only, our proof does not directly concern us with the cardinality of the Real numbers. We are currently only concerned with the foundations that lead to these prior results, and the counterexamples which call them into sincere question.

We will re-explore CDA by creating an Abelian group which can be used as a representation of an \(\omega\)-regular language, and its set of well formed strings, explore this representation’s properties and expressive power as a language, and create a class of constructions of CDA which lead to non-trivial counterexamples to CDA. The resulting counterexamples to CDA are found through a transformation of equivalent statements over the construction.

### 2.2 Representing an \(\omega\)-language by a Recursively Enumerable Language

#### 2.2.1 Notation and Preliminaries

**Definition** Let the Natural numbers be denoted by \(\mathbb{N}\).

**Definition** Let \(\omega\) be the rank of \(\mathbb{N}\), from the Von Neumann Hierarchy.

**Definition** Let \(\aleph_0\) be the cardinality of \(\mathbb{N}\) and the cardinality of any set bijective to \(\mathbb{N}\).
Definition An \(\omega\)-expansion is an unbounded expansion of symbols on a string \(\text{Lim} \rightarrow \omega\), and is denoted, for some symbol, \(z\), as \(z^\omega\) such that the cardinality of the set of the number of symbols represented by the \(\omega\)-expansion \(z^\omega\) is \(\aleph_0\).

Definition An iteration of symbols has gone through \(\omega\)-completion when an unbounded set of recursions of symbols in a string has a symbol or the empty string, at the rank of the set of iterations.

Definition Let exhaustion be a necessary change in output of a recursion after \(\omega\)-completion of such recursion of symbols, where \(s\) at exhaustion is the symbol at the rank of the unbounded iteration and \(s \neq i\) when \(i\) is the first symbol of the set of symbols in the recursion.

For example, let \(x\) be defined as an infinite word having both 0 and 1's written on an infinite tape, however, we know that the starting symbol is 1, and there are an \(\omega\) number of 1s on the tape prior to any 0 appearing. We can prove by exhaustion that the \(\omega^{th}\) symbol of the infinite word is 0, since \(\omega\) is the rank of the total number of 1s on the infinite tape and the symbol must change necessarily at that point, as we know the word contains a 0.

Definition Let a special order of some set \(S\) be some ordering of \(S\) such that we may determine the value of any distinct element of \(S\) by some operation on \(S\) which guarantees the result is in \(S\); i.e. an order which is arbitrary under closure.

Definition Let \(\Sigma\) be an alphabet with at least 6 elements and the empty string \(\epsilon\).

Definition Let \(\Sigma^\omega\) be the set of all strings over \(\Sigma\) which includes all \(\omega\) length strings.

Definition Let \(\Sigma^\omega_2\) be the set of strings which also include strings of length \(\omega\) over a subset \(\Sigma_2\) of at least two elements in \(\Sigma\).

Definition Let \(\Sigma^\omega_3\) be the subset of \(\Sigma^\omega_2\) which are only the recursive, countably infinite length strings over \(\Sigma_2\) so that the set of all symbols in each string has a cardinality \(\aleph_0\).

Definition A language of the class of computable transfinite number \(T\) has a subset of unique finite length strings \(\Sigma^*_4\) of \(\Sigma^\omega_3\) that is either bijective or surjective (but not injective) to the set \(\Sigma^\omega_3\).

Definition The alphabet \(\Sigma_4\) must also have a subset of at least three elements not a member of \(\Sigma_2\) with at least one of these symbols to distinguish context between other symbols of value.

Remark Note that the total cardinality of the set of strings in \(\Sigma^\omega_3\) is \(\aleph_0^2\) and fully representable by an \(\Sigma^*_4\) mapping via Cantor pairing between the sets.

Definition Let \(\Sigma^*_2\) be the set of finite length strings over \(\Sigma_2\). For any language \(L\) in the class \(T\), the set of strings \(\Sigma^*_L\) over \(L\), \(\Sigma^*_L = \Sigma^*_4 \cup \Sigma^*_2\).

### 2.2.2 A Grammar for a Language in \(T\)

Chomsky, Backus and Naur laid the foundation for work in generative grammars and our understanding of computer syntax with Backus-Naur form and the Chomsky hierarchy among other means of representing and categorizing generative grammar structures. Here we utilize formal language theory to produce the following grammar which
yields a recursively enumerable context sensitive language.\[3\]

**Definition** Let the grammar $L_T(G)$ be a tuple, \{$V, \sum, R, S\}$. $S \in V$, $\sum_2 \subset \sum$, $\sum_3 \subset \sum$, such that:

$V \cap \sum = \emptyset,$

$V=\{S, T, A, B, C, Q, U\},$

$\sum_\varepsilon=\{\varepsilon, 0, 1\},$

$\sum_2=\{0,1\},$

$\sum_3=\{0,1\},$

$\sum_4=\{\varepsilon, 0, 1, [\, , ]\},$

$R=\{S \rightarrow \varepsilon, S \rightarrow T, T \rightarrow \varepsilon, T \rightarrow A, T \rightarrow B, T \rightarrow C, T \rightarrow Q, A \rightarrow 1, A \rightarrow 0, A \rightarrow 0A, A \rightarrow 1A, A \rightarrow 0C, A \rightarrow 1B, A \rightarrow 111, B \rightarrow \varepsilon, B \rightarrow 0, B \rightarrow 0A, B \rightarrow 000, B \rightarrow 000, B \rightarrow 000, C \rightarrow 1, C \rightarrow 1A, C \rightarrow 1B, C \rightarrow 111, Q \rightarrow [Q], Q \rightarrow [U], Q \rightarrow T[U]T, Q \rightarrow T[U]T[U]T, U \rightarrow 01B, U \rightarrow 10C, U \rightarrow 0A, U \rightarrow 1A, U \rightarrow 000C, U \rightarrow 111B\}$

**Definition** Let $L_T$ be a language of the elements in $\sum_4^*$ which generate only from grammars generated by the non-terminals $B, C, Q$ and $U$ that cannot be generated exclusively by the non-terminal $A$. Let the elements of $\sum_4^*$ have an onto mapping to the recursive strings of infinite length in $\sum_3^\omega$ (i.e. those strings from $\sum_3^\omega$ which are readable by B"uchi automaton). \[1\]

**Definition** We define $\rho^*$ as the set of all well formed strings over $L_T$.

*Lemma* $\rho^*$ is an Abelian Group

**2.2.3 Addition in $\rho^*$**

**Definition** $\text{LENGTH}(x)$ is a function which outputs the length of a string within brackets.

**Definition** $\text{PLUS1}(w) = w + 1 = 1 + w$ and follows these rules:

$\text{PLUS1}(0) := 1$

$\text{PLUS1}(1) := 10$

$\text{PLUS1}([1]) := [1]$

$\text{PLUS1}(0) := 0$

$\text{PLUS1}([0]) := [0]$

$\text{PLUS1}(01) := [01]$

$\text{PLUS1}(10) := [10]$

$\text{PLUS1}(10) := [101]$

$\text{PLUS1}(01) := [010]$

$\text{PLUS1}(10) := [100]$
Definition \( \text{PLUS}(w) = w + 1 = 1 + w \) and follows these rules:

\[
\begin{align*}
\text{PLUS}(0) & := 1 \\
\text{PLUS}(1) & := 0 \\
\text{PLUS}(1) & := 10 \\
\text{PLUS}(0) & := 1 \\
\text{PLUS}(10) & := [10]01 \\
\text{PLUS}(01) & := [01]00 \\
\text{PLUS}(10) & := [10]1 \\
\text{PLUS}(01) & := [10] \\
\text{PLUS}(10) & := [10]1
\end{align*}
\]

Definition Let \( \text{PLUS}() \) be commutative such that \( \text{PLUS}(w_1, w_2) = w_1 + w_2 = w_2 + w_1 \), and follows these rules:

\[
\begin{align*}
\text{PLUS}(w_10, w_20) & := \text{PLUS}(w_1, w_2)0 \\
\text{PLUS}(w_11, w_20) & := \text{PLUS}(w_1, w_2)1 \\
\text{PLUS}(w_10, w_21) & := \text{PLUS}(w_1, w_2)1 \\
\text{PLUS}(w_11, w_21) & := \text{PLUS}(w_1, w_2)10 \\
\text{PLUS}(w_10, w_20) & := \text{PLUS}(w_1, w_2)0 \\
\text{PLUS}(w_11, w_20) & := \text{PLUS}(w_1, w_2)0 \\
\text{PLUS}(w_10, w_21) & := \text{PLUS}(w_1, w_2)0 \\
\text{PLUS}(w_11, w_21) & := \text{PLUS}(w_1, w_2)01
\end{align*}
\]

Remark Similarly, brackets may also appear in \( w_2 \), but details are omitted for the sake of brevity.

Definition To define the set \( P^* \), let us suppose Kleene star expansions of strings
over $\rho^*$: ‘$(w_1)^*[(w_2)^*]^{*}(w_3)^*$’ and an expansion rule such that when there exists the terminal $\emptyset$ in a string, there can be a Kleene star expansion of ‘0’ either to the left or the right or both of $\emptyset$ such that ‘$0^*\emptyset0^*$’ is a valid string and similarly for 1 such that ‘$1^*11^*$’ is a valid string. $P^*$ is the union of $\rho^*$ and all such Kleene star expansions in $\rho^*$.

**Definition** Let an $\omega$-expansion of a string be an infinite iteration operator similar to a Kleene star expansion, but for infinite words.

**Definition** Let $\text{LENDIFF}(\cdot)$: For all $x$ in strings $w_3, w_4$:

\[
\text{LENDIFF}(\text{LENGTH}(x_1)^*\text{LENGTH}(x_2)) := \text{LENGTH}(x_y) \text{ where } x_y \text{ is a concatenation of } x_1 \text{ LENGTH}(x_y) \text{ times.}
\]

\[
\text{LENDIFF}(\text{LENGTH}(x_1)^*\text{LENGTH}(x_2)) := \text{LENGTH}(x_z) \text{ where } x_z \text{ is a concatenation of } x_2 \text{ LENGTH}(x_z) \text{ times.}
\]

When $\text{LENGTH}(x_1) \neq \text{LENGTH}(x_2)$:

**Definition** $\text{PLUS}(w_1[x_y], w_2[x_z]) := \text{PLUS}(w_3, w_4)$

**Remark** Addition is Commutative: By the definition of the PLUS function, addition is commutative in $\rho^*$; the equation $a + b = b + a$ holds for all $a, b$ in $\rho^*$.

**Remark** It is easy to allow addition to be associative in $\rho^*$ such that the equation $(a + b) + c = a + (b + c)$ holds for all $a, b$ and $c$ in $\rho^*$.

**Remark** The identity element exists as $0$, for all $w$:

\[\text{PLUS}(w, 0) = w\]

**Definition** Let *String Equality* be the condition where expanded strings from a formal language, either by Kleene star expansions or $\omega$-expansions, have equality with strings from which they expanded.

**Definition** Let $w_1$ be a string from $\rho^*$ and let $w_2$ be a string expanded from $w_1$ such that for every symbol $\emptyset$ in a string, there is string equality when $\emptyset = \emptyset^0$ and $\emptyset = \emptyset^{0^\omega}$. There is also string equality when for every symbol $1$ in a string, $1 = 11^*$ and $1 = 11^\omega$.

**Definition** Let $P^\omega$ be all the strings in $\rho^*$ union all $\omega$-expansions and Kleene star expansions of symbols in $P^*$. Because of string equality, for $w_1$ in $\rho^*$ and for $w_2$ in $P^\omega$ $\text{PLUS}(w_1, 0) := \text{PLUS}(w_2, 0)$.

**Definition** Let $P^\prime$ be all strings in $P^\omega$ which are not in $\rho^*$.

**Proposition** Addition in $\rho^*$ is closed. $\text{PLUS}(\cdot)$ will only yield answers in $\rho^*$ or $P^\prime$.
by the function PLUS( ). For any strings \( w_1 \in \rho^* \), \( w_2 \in \rho^* \), \( \text{PLUS}(w_1, w_2) := w_a \), if \( w_a \in P \), there exists a String Equality in \( \rho^* \), such that \( w_a = w_b \) where \( w_b \) is in \( \rho^* \) and \( \text{PLUS}(w_1, w_2) := w_b \) through reflexivity. Since \( w_b \notin P \), it must be in \( \rho^* \). Therefore, addition is closed in \( \rho^* \).

**Proposition** There is an additive inverse for each \( w \in \rho^* \). For every \( w_x \), there exists some \( w_y \) through iteration in \( \text{PLUS}(w_x, w_y) \) such that the output string will be some combination of terminals \( 0^* \) and \( 000^* \), which through String Equality\(=0 \).

**Lemma.** \( \rho^* \) is an Abelian Group under addition. From the previous two propositions, and because or definition of \( \text{PLUS}( ) \) includes commutativity, by the definition of an Abelian Group as having closure, associativity, an identity element, an inverse element and commutativity, \( \rho^* \) is an Abelian Group under addition. \( \square \)

### 2.3 A Non-trivial Counterexample to Cantor’s Diagonal Argument

#### 2.3.1 Formalizing a Diagonal Argument Function

For Cantor’s Diagonal Argument, one may generalize an argument function, for proof by contradiction, through constructing an arbitrary sequence of infinite length strings in any order into an \( N \) by \( N \) matrix of symbols and assuming all rows of the matrix contains all string values. Then, one proceeds to the \( i^{th} \) column, \( j^{th} \) row of the matrix, where \( (i, j) \) moves along the diagonal of the matrix- \( (1,1), (2,2), (3,3) \)… etc. The symbol at that position is then changed to a different symbol within the language of the system and concatenated to produce a new string, which when so constructed, the new string of symbols can’t be found in any row or column of the matrix, thus providing a string of a value not listed in the matrix. The new string is considered transcendental in Cantor’s Universe and the proof by contradiction tells us that not every string value can be calculated in Cantor’s Universe through iterative process, and as such, \( \mathbb{R} \) is “uncountable” and has a cardinality strictly larger than \( \mathbb{N} \). This is necessarily true for all constructions of the diagonal argument prior to the method employed in this paper because the \( i^{th} \) string must contain the symbol found at \( (i, j) \) which cannot be in the constructed string at \( (i, j) \) from the diagonal and there did not yet exist a mapping relationship suitable for counterexample to CDA. \[3\]

The following construction maps the set \( \rho^* \) with equivalent strings in \( P^\omega \), showing that the diagonal argument applied to a special ordering of subsets in \( P^\omega \), yields a non-trivial counterexample to Cantor’s argument function.

**Remark** It should be noted that because the elements of \( P^\omega \) are not the Real numbers, and thus, the trivial exception where \( .9=1 \) is not a concern because this evaluation or any equivalent evaluation does not exist in \( P^\omega \) by construction and this exception simply cannot occur. We will not concern ourselves with this exception, since it is a special case, it is considered trivial. However, if such a case could be generalized, it would no
longer be trivial.

2.3.2 Additional Notations and Preliminaries

Consider a proof by contradiction as an argument function \( f(S) \) over a set \( S \) yielding a domain \( f(S) \rightarrow g(S) \) and some countable set of matrices reducible via Cantor Pairing to size \( \mathbb{N} \times \mathbb{N} \), \( \{ f(S) \} \), to which we will apply an argument function \( f(S) \).

**Definition** A *counterexample* to an argument function (which is a proof by contradiction), is the condition when there exists \( f(S) \rightarrow g(S) \) such that for any \( g(S) \in \{ f(S) \} \) iff \( g(Q) \notin \{ f(Q) \} \) OR if any \( g(S) \notin \{ f(S) \} \) iff \( g(Q) \notin \{ f(Q) \} \) for sets \( S \) and \( Q \).

**Definition** Let right-concatenation be the concatenation operator \( * \) such that when \( w_1 \) right concatenates over \( w_2 \), \( w_1 * w_2 = w_2 w_1 \) and likewise, \( w_1 * w_2 * w_3 * w_4 = w_4 w_3 w_2 w_1 \).

**Definition** Let \( \text{str}(f(x_j)) \) be the cumulative string of the output of the function \( f(x_j) * f(x_{j-1}) * f(x_{j-2}) \ldots * f(x_{j-j}) \). Similarly for \( \text{str}(g(x_j)) \).

**Definition** Let the argument function \( f(\mathbb{R}) \) be a construction of Cantor's diagonal method on the \( \mathbb{N} \times \mathbb{N} \) matrix \( \{ f(\mathbb{R}) \} \) of arbitrary unbounded binary strings. They are listed in any order, assuming that all possible string values of \( \mathbb{R} \) are listed. Let \( g(\mathbb{R}) = \text{str}(g(x_j)) \) where \( x_j \) is the value, 0 or 1 at the coordinate \((i,j) \) for all \( j \) on the constructed matrix at row \( j \) such that \( j = i \). It is well established that in Cantor's Universe, \( g(\mathbb{R}) \notin \{ f(\mathbb{R}) \} \), and we can produce a string which is in \( \mathbb{R} \) but not in \( \{ f(\mathbb{R}) \} \).

For counterexample, since we accept \( g(\mathbb{R}) \notin \{ f(\mathbb{R}) \} \), we must find some set of recursively enumerable sets or set of uncountable sets \( S \) where \( g(S) \in \{ f(S) \} \).

**Definition** Let the argument function \( f(P_0^\omega) \) be an equivalent construction of Cantor's diagonal method where a series of \( \mathbb{N} \times \mathbb{N} \) matrices, \( \{ f(P_0^\omega) \} \), is partitioned into two specially ordered matrices, \( \{ f(P_1^\omega) \} \) and \( \{ f(P_2^\omega) \} \). \( \{ f(P_1^\omega) \} \subset \{ f(P_0^\omega) \} \) and \( \{ f(P_2^\omega) \} \subset \{ f(P_0^\omega) \} \). These two partitions will be proper disjoint subsets of \( \{ f(P_0^\omega) \} \) such that \( \{ f(P_1^\omega) \} \cup \{ f(P_2^\omega) \} = \{ f(P_0^\omega) \} \).

Construct the argument function for application of CDA to \( P_0^\omega \), \( \{ f(P_0^\omega) \} \), in the following manner: Let \( \rho^* \) map to two sets, \( \rho_1^* \) and \( \rho_2^* \), through choice, choosing the strings in \( \rho_1^* \) as all the strings where each string is denoted by the initial with Kleene star expansions, from left to right as follows: \( 0w, \theta w, [^*0w \text{ or } \theta w \]. And likewise, choose the strings in \( \rho_2^* \) as all the strings where each string is denoted by the initial, from left to right, \( 1w, \text{idw, } [^*1w \text{ or } \text{idw, thus creating a strictly bijective disjunction between the sets whose union is } \rho^*. \) Choose some string set in \( \rho_1^* \) such that it maps to an infinite length string by String Equality in \( P^\omega \) to form the set \( P_1^\omega \). Choose some string set in \( \rho_2^* \) such that it maps to some infinite length string by String Equality in \( P^\omega \) to form the set \( P_2^\omega \). Let the union of \( P_1^\omega \) and \( P_2^\omega \) be \( P_0^\omega \).

We will begin the proof of the existence of the counterexample to CDA by proceeding
with a construction of $f(P_{1}\omega)$ and $f(P_{2}\omega)$ individually, and then providing a retrograde on $str(g(P_{2}\omega))$ to construct the diagonal of $\{f(P_{0}\omega)\}$.

2.3.3 Theorem

Proof. There is a Non-trivial Counterexample to Cantor’s Diagonal Argument.

1. $\rho^\ast$ is an Abelian group by Lemma.
2. By String Equality we can create an Abelian Relationship in all extended sets related to $\rho^\ast$ by providing an operation on a string in $\rho^\ast$ to yield another result in $\rho^\ast$, which can then be made equal to some string in $P\omega$ and its subsets, thus allowing for special order, utilizing exhaustion, if necessary.
3. We will choose the elements of $P_{1}\omega$ and $P_{2}\omega$ in such a way, that when proceeding with the argument function down the diagonal of the respected matrices, $\{f(P_{1}\omega)\}$ and $\{f(P_{2}\omega)\}$, we will encounter only the symbols 0 and 1, respectively. This can be ensured by expanding other symbols when necessary. We will reserve the axiom of replacement if necessary when forming the sets to aid in our choice for the rows in the respective matrix sets.
4. Proceeding to $f(P_{1}\omega)$, we choose to write the row, $j=1$, as some 01 as it’s defined mapped string, 011$\omega$ and by the argument function, utilize the symbol from the 1st column of this row such that $f(x_{1})$:=0 and $str(g(x_{1}))$:=1.
5. Choose in the set such that the second row in $\{f(P_{1}\omega)\}$ is 011$\omega$=10$\omega$1$\omega$1 and $f(x_{2})$:=0, and $str(g(x_{2}))$:=11 in accordance with the argument function.
6. Thus, through $\omega$-completion of the recursive iterations resulting in an unbounded series, for all $j$ in $\{f(P_{1}\omega)\}$, $f(x_{j})$:=0; $g(x_{j})$:=1.
7. It follows through String Equality that $g(P_{1}\omega)$=1.
8. Likewise, proceeding to $f(P_{2}\omega)$, we choose to write the row, $j=1$, 10 as it’s defined mapped string as some 100$\omega$ and by the argument function, utilize the symbol from the 1st column of this row such that $f(x_{1})$:=1 and $str(g(x_{1}))$:=0.
9. Choose in the set such that the second row in $\{f(P_{2}\omega)\}$ is 101$\omega$=10$\omega$0$\omega$0 and $f(x_{2})$:=1, and $str(g(x_{2}))$:=00 in accordance with the argument function.
10. Thus, through $\omega$-completion of the recursive iterations resulting in an unbounded series, for all $j$ in $\{f(P_{2}\omega)\}$, $f(x_{j})$:=1; $g(x_{j})$:=0.
11. It follows through String Equality that $g(P_{2}\omega)$=0.
12. The retrograde of $g(P_{2}\omega)$=0 is reflexive, 0.
13. The special order of $\{f(P_{0}\omega)\}$ is the special order of $\{f(P_{1}\omega)\}$ union the retrograde of the special order of $\{f(P_{2}\omega)\}$, allowing for an inversion on the strings in the diagonal, because of retrograde, for consistency in the argument function.
14. Through String Equality, $g(P_{2}\omega)$=10
15. By choice of $\{f(P_{2}\omega)\}$ which is a subset of $\{f(P_{0}\omega)\}$, as illustrated by step 9 in this proof, the string 101$\omega$0$\omega$0 is in $\{f(P_{0}\omega)\}$.
16. Therefore, there exists a counterexample; when $g(\mathbb{R}) \notin \mathbb{R}$, $g(P_{0}\omega) \in P_{0}\omega$. 


2.4 Discussion

We can easily generalize this counterexample for all elements in \( \rho^* \) by choosing the appropriate ordering of the list of elements used for constructing the diagonal.

Finding such a non-trivial counterexample to CDA provides a new understanding about the limits of computability and these results confirm the flaw found in the Entscheidungsproblem. It would be wise for logicians, mathematicians and computer scientists to revise all their texts accordingly. However, I recognize that this is quite an ordeal, so we will venture one step further, beyond what would be required of any other paper or result, and provide a third argument as it relates to Gödel’s Diagonalization Lemma, which is a third method that differs from the previous two already discussed, but is reducible to the same result.

3 Gödel’s Diagonalization Lemma is Not Logically Contingent

3.1 Some Impredicative Statements Reduce to Circular Logic

An impredicative statement is a statement with self-referencing. Self-reference alone is not enough to invalidate a proposition, however, as we will show, a certain class of impredicative statements are intrinsically circular and tautological. Such circular logic is not well-formed for proof of a proposition which depends on its circular nature, and is thus not logically contingent.

In this section, we will show that Gödel’s Diagonalization Lemma is within this class of impredicative statements. We will then review this lemma and show that his utilization of substitution is not strict enough for a strong foundation in logic. Furthermore, we will actually construct the Gödel numbers suitable for his proof, and carry out the operations to see if indeed such arithmetic is outside of a formal system, \( Q \).

First, we must determine a means to identify when a tautology is present in an impredicative definition. Let’s start with an example of a tautology. We would not accept the following as a logical statement as true in any meaningful way (it is, in fact, "true" semantically), because there is no proof or causal relationship, only tautology:

\[(S \lor \neg S)\]

We can determine if a statement is a tautology or not, by creating a truth table over the distinct valuations for the formula. If all valuations for each variable lead to truth, then the statement is a tautology.

For \((S \lor \neg S)\), When S is true, \((S \lor \neg S)\) is true. When S is false, \((S \lor \neg S)\) is true. Thus, \((S \lor \neg S)\) is a tautology.

So what about impredicativity? Intuitively, it is easy to see how we risk creating a tautology by using such a tactic to define one’s terms: Self reference intrinsically eliminates the possibility for that variable to contradict itself, increasing the likelihood of tautology.

However, self-reference or impredicativity alone, is not enough to prove tautology. For example, the greatest lower bound (glb) of a set, can be defined impredicatively, \( y = \text{glb}(X) \) if and only if for all elements \( x \) of \( X \), \( y \) is less than or equal to \( x \), and any \( z \)
less than or equal to all elements of X is less than or equal to y. This definition is impredicative, but it is not tautological.

But now, intuitively, before we prove for GDL, I would like to claim that Russell’s paradox uses an impredicative definition that is itself actually tautological.

Define S as the set of all sets that do not contain themselves. This is impredicative, and it is tautological, because there seems to be this infinite self-referencing that goes on when we apply S to itself.

So what differentiates these two classes of impredicative definitions from each other? Heuristically, in the first instance, we are using the set of X to define y, which is the greatest lowest bound of X. In the second instance, we are defining S as a set defined by sets. When we apply the distinguishing property to the definition that makes the term unique, in the instance of the greatest lowest bound, the property is not applied to the impredicative portion of the statement. Rather, the property is applied to an independent portion. That is, the property of being an element is distinguished from the property of being a set, the set, in this instance, being the impredicative portion of the statement.

In Russell’s paradox, the property of being a set that does not contain itself, is a property applied to a dependent portion of the impredicative statement, that is, the property when applied to sets in general, also applies to the specific set we are defining. This dependence of the property of the set on the set, is the distinguishing factor. I believe it was Wittgenstein, at a dinner party with Russell, who first noted that Russell’s paradox is easily overcome by defining a class of sets that do not contain themselves. When you define a class of sets, you break the dependence of it as a set (since it is no longer a set, but a class) with given properties that apply to sets.

But I believe this heuristic falls short. It is not just this dependence that creates a tautological impredicative statement, it is also the nature of the property itself. We could easily define the set of all sets which contain the letter A. Such a definition has a dependence on impredicative portion of the statement, yet does not seem to create a tautology or an infinite regression or anything of that sort. That is, in order to create a tautology, A itself would have to be defined, not only in terms of sets, but in terms of the class of sets in question. As such, in order for impredicatives to be a problem for logic, the property itself must point back to the impredicative dependence in a self referential way.

**Definition** Letimpredicative dependence be the condition of a statement S, whose property P depends on self-referencing. \( \exists x | P(x) \leftrightarrow \{ x \rightarrow P(x) \} \)

**Definition** Letimpredicative pointing be a condition of self-reference where a dependent property also references an impredicative dependence, i.e. the existence of S depends on S containing an impredicative dependence. \( \exists x | P(S) \leftrightarrow P(x) \leftrightarrow \{ x \rightarrow P(x) \} \rightarrow S \)

**Definition** Let a tautological impredicative be a statement which satisfies impredicative pointing. \( \exists x | P(S) \leftrightarrow P(x) \leftrightarrow \{ x \rightarrow P(x) \} \rightarrow S \)

**Proposition** Tautological impredicatives are logical tautologies. \( S \models \exists x | P(S) \leftrightarrow \{ x \rightarrow \)
The final statement S is always true, even when S is assumed to be false.

|   | x | P(S) ↔ \{x → P(x)\} → S |
|---|---|--------------------------|
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | true |

3.2 Gödel’s Diagonalization Lemma Depends on a Tautological Impredicative

Gödel’s Diagonalization Lemma states that there is a sentence Ψ such that Ψ ↔ F(o(#(Ψ))) is provable in Q where o(#(Ψ)) is the Gödel number for Ψ and F is some well formed formula provable in a formal system Q.

It is clear to see that the statement concerning Ψ is a tautological impredicative, as Gödel derives Ψ by the following proof.

3.2.1 Gödel’s Diagonalization Lemma

**Proof:** \(Q \vdash Ψ \leftrightarrow F(o(#(Ψ)))\).

Given a formula with one free variable, F(x) in Q and a number n, we may substitute some number n for x. We may also represent the formula F(x) by the Gödel number of the formula, \(o(#(F(x)))\) and of a number n, and that it is possible that \(n := o(#(F(n)))\). We can refer to this process of substitution by the function substn\((o(#(F(x))), n) := o(#(F(n)))\). We can also let \(S(x, y, z)\) be a formula which strongly represents this operation in the language of Q if and only if:

\[x = o(#(F(x))), y = n, \text{ and } z = o(#(F(n)))\]

Nothing prevents \(y := x\), such that we may have the formula \(S(x, x, z)\).

Given any formula F(x), we may also create the formula \(\exists z[F(z) \land S(x, x, z)]\) with one free variable, x. This formula has the Gödel number \(k := o(#(\exists z[F(z) \land S(x, x, z)]))\). At this point, we may substitute k for x, such that \(\exists z[F(z) \land S(k, k, z)]\). Let \(Ψ := \exists z[F(z) \land S(k, k, z)]\).

Let \(m := o(#(\exists z[F(z) \land S(k, k, z)]))\). \(Q \vdash \forall z[S(k, k, z) \leftrightarrow z = m]\).

\(Q \vdash \forall z[S(k, k, z) \leftrightarrow z = o(#(\exists z[F(z) \land S(k, k, z)]))\]

\(Q \vdash Ψ \leftrightarrow F(o(#(\exists z[F(z) \land S(k, k, z)]))\) and from this...

\(Q \vdash Ψ \leftrightarrow F(o(#(Ψ)))\) \[6\]

**Remark.** Bew() is short for the german word beweisbar, which means “provable”. Note that “Bew(x)” is merely an abbreviation that represents a particular, very long, formula in the original language of Q; the string “Bew” itself is not claimed to be part...
Ok, fine. This seems logical, but what happens when \( F(x) := \neg \text{Bew}(x) \)?

We get \( Q \vdash \Psi \iff \neg \text{Bew}(\#(\Psi)) \). This is a contradiction. The proper question, given the previous two section in this article, is to ask, why do we have such a contradiction arise? Is it because of incompleteness, or is it a result which only exists because of a fundamental flaw in the diagonalization lemma, which creates a logical tautology?

I wish to point out, that if \( \neg \text{Bew}(x) \) applies to \( \Psi \), then, this is the function which must also apply to \( z \).

Such that \( \neg \text{Bew}(\Psi) \leftrightarrow \neg \text{Bew}(z) \), where:

\[
\begin{align*}
Q \vdash \Psi \leftrightarrow \neg \text{Bew}(\#(\exists z[\neg \text{Bew}(z) \land S(k, k, z)]))
\end{align*}
\]

which means that the fundamental assumption we made about there existing a function \( F(x) \) with one free variable, is what is incorrect... that it is through improperly initiating substitutions, when we create tautology, which allows contradiction. This is a fundamental problem with the logic used in Gödel’s proof, not necessarily \( Q \). There could be some \( Q \) where stronger foundations in logic is mandatory. Furthermore, we can easily see that \( Q \) allows tautological impredicatives by proving \( \Psi \).

### 3.2.2 \( Q \vdash \Psi \iff F(\#(\Psi)) \) is a Tautological Impredicative

Let the formal system \( Q \) be represented by the statement \( S \) in \( Q \).

Let \( \#(x) \) be the property \( P(x) \), such that

\[
Q \vdash \Psi \leftrightarrow F(\#(\Psi)) := S \vdash \Psi \leftrightarrow F(P(\Psi)),
\]

since \( k \leftrightarrow P(x) \),

and \( z \leftrightarrow k \rightarrow P(P(x)) \),

then \( P(x) \leftrightarrow P(P(x)) \rightarrow z \), which proves impredicative dependence, our first condition of tautological impredicativity.

If \( \Psi \leftrightarrow \exists z[F(z) \land S(k, k, z)] \), and \( P(x) \leftrightarrow P(P(x)) \rightarrow z \),

then \( \Psi \leftrightarrow \{P(x) \rightarrow P(P(x))\} \rightarrow z \), thus,

\[
\begin{align*}
P(z) &\leftrightarrow \{P(x) \rightarrow P(P(x))\} \rightarrow \Psi, \text{ and since} \\
\Psi &\leftrightarrow z,
\end{align*}
\]

\( P(\Psi) \leftrightarrow \{P(x) \rightarrow P(P(x))\} \rightarrow \Psi \) which satisfies our definition of a tautological impredicative, which means that GDL is a logical tautology and his proof can not be logically contingent.
3.2.3 Discussion

Did Gödel overstep in his logic by taking too many liberties with free variables?

The answer to such a question, is perhaps a matter of opinion. It is either OK to prove the existence of statement which is a logical tautology in a formal system, or it is not. If it is OK, I think we would have to also assume it would be difficult to prevent such a tautology from occurring, or that tautologies must be desirable in logic. However, because tautologies are not desirable, as they open up systems to paradox, meaninglessness and contradiction, I believe it would be of a higher opinion to seek out a rule which invalidates their existence in a consistent system; that any system which can prove tautologies, is in fact inconsistent.

Such a rule is relatively easy to find. It seems, unless one can offer an additional example than the one provided by Gödel, that tautologies in a formal system such as $Q$, are only made possible by misusing substitution over a free variable. Substitution, being an important foundational element in logic, is not itself in question. But rather, a limit on how substitution may be applied, perhaps should be.

We could just create a postulate or axiom which makes $x$ a bounded variable after substitution.

Such a postulate will allow some impredicative statements, all of which can avoid tautology, but through preventing re-substitution on $x$, will prevent impredicative pointing, and thus prevent tautologies from forming in $Q$. Thus, creating a much stronger foundation to our systems of logic and computability.

Or, if such a postulate is too restrictive, we can modify the limit of substitution over bounded $x$, not to substitution in general, but to statements from the free variable $x$, formally as follows:

**Axiom.** For any formal system $Q$, with a free variable $x$ in formula $A(x)$, after substitution is applied to $x$, $x$ is bounded by the substitution function such that $\text{subst}(\text{subst}(x)) \neq \text{subst}(x)$.

Such a rule will strengthen, not weaken the logical prowess of humanity.

4 Conclusion, P=NP

While only one method was ever truly necessary, by three novel, independent and rigorous methods, we have been able to deconstruct one of the key foundations of mathematics and computer science, diagonalization. In Section 1, we constructed an impossible Supermachine which should not exist, yet it does. In section 2, we constructed a formal language which yields a non-trivial counterexample to Cantor’s Diagonal Argument. And in Section 3, we tackled the impredicative nature of Gödel’s diagonalization lemma, revealing a tautology, which is logically unfounded. Again, any one method should have been enough for proof, but because of the significance of such a finding, I decided to combine over a decades worth of research and original thought into this single cohesive, self-contained paper.
Finally with diagonalization methods invalid, the entire complexity hierarchy collapses, and we now have a foundation in computer science where there is enough information to solve the P vs. NP problem. Without diagonalization, our reasons for not using an oracle to solve P vs. NP vanish, as the contradictions which formally prevented the use of such oracle no longer exist. It was inconsistency, through the use of diagonalization that lead to hierarchy results which created oracle contradictions.

\[ \text{Lemma.} \quad \text{If } PSPACE = EXPSPACE, \ P = NP. \] If the SPACE of a problem increases polynomially as with any PSPACE-complete problem, this is comparable to the TIME of a problem increasing polynomially, such that given an oracle, \( =_{\text{poly}} \), which solves polynomial equivalence between SPACE and TIME, \( PSPACE =_{\text{poly}} P \).

Similarly, if the SPACE of a problem increases exponentially as with any EXPSPACE-complete problem, this is comparable to NP which is at maximum, in exponential TIME, such that \( EXPSPACE \geq_{\text{poly}} NP \). If \( PSPACE = EXPSPACE \), then \( PSPACE \geq_{\text{poly}} NP \). Since \( PSPACE =_{\text{poly}} P, \ P \geq_{\text{poly}} NP \), which is the same as \( P = NP \).

Solving for \( \beta' \) in Section 1 is RE complete, and because the Space Hierarchy Theorem relies fully on the now defunct diagonalization method, its results must be discarded.

And as such, with the Entscheidungsproblem being RE-complete, and since we may solve for arbitrary \( \beta' \) using the Supermachine configuration, we may now conclude:

\[ \text{Proof.} \quad \text{Since by definition, } PSPACE \subseteq RE, \text{ and} \]

since any given Recursively Enumerable set is contained in PSPACE, and \( \beta' \) solves for all Recursively Enumerable sets in PSPACE, and since we can no longer accept the Space Hierarchy Theorem, \( RE \subseteq PSPACE \ldots \)

\( RE = PSPACE, \)

such that \( EXPSPACE \subseteq RE \)

and \( RE = PSPACE, \) implies \( PSPACE = EXPSPACE, \) proves through the above Lemma ...

\( P = NP \)

Mark Inman, Ph.D.

Prague, Czech Republic

References

[1] Büchi, J.R., On a decision method in restricted second order arithmetic. Proc. International Congress on Logic, Method, and Philosophy of Science. 1960, pages 1–12, Stanford, CA, Stanford University Press. 1962

[2] Lewis, Harry Introduction to Formal Systems and Computation CS121 Harvard College December 2007 [http://www.seas.harvard.edu/courses/cs121]
[3] Sipser, M. *Introduction to the Theory of Computation, 3rd ed.*, Cengage Learning, Boston 2013

[4] Staiger, L., ω-*Languages* G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, Volume 3, pages 339-387. Springer-Verlag, Berlin, 1997.

[5] Suppes, Patrick *Axiomatic Set Theory* Mineola, NY: Dover Publications, Inc. 1972

[6] https://plato.stanford.edu/entries/goedel-incompleteness/sup2.html

[7] A. M. Turing “On Computable Numbers, with an Application to the Entscheidungsproblem” 1936-1937: Proceedings of the London Mathematical Society, Series 2, 42, pp 230-265.

[8] https://en.wikipedia.org/wiki/Impredicativity

[9] https://en.wikipedia.org/wiki/Gödel's_incompleteness_theorems