The Julius Silver, Roslyn S. Silver, and Enid Silver Winslow
DIALOGUES IN ARTS AND SCIENCE

THE CESSATION OF CERTAINTY

Joel Spencer
Silver Professor of Mathematics and Computer Science

NEW YORK UNIVERSITY

The Cessation of Certainty

> You just keep right on thinking there Butch.
> Thats what you're good at.
>
> Robert Redford to Paul Newman, *Butch Cassidy and the Sundance Kid*

The 20th century saw the introduction of *undecidability* into our mathematical language. It appeared originally in the work of Gödel. Here, however, the issue is approached from a Computer Science vantage point. This approach has technical validity. More importantly, in our current century with computers so ubiquitous the notions needed, most particularly that of algorithm, are widely known (even if not totally understood) through- out the academic community.

1. Ceci n'est pas une pipe

The signifier and the signified are usually clearly distinguished. From the art of Magritte and Escher to the writings of Auster their conflation fascinates us, leading to puzzlement and paradox. Here is a semantic version.

Some words describe themselves. Short is short and sesquipedalian is sesquipedalian. But long is not long and red is not red. Here we have a dual use

(or confusion, if you will) between the notion of word as something that describes things and the notion of a word as the object, something that may be described. Call a word selfreferential if, like short and sesquipedalian, it describes itself; otherwise call it nonselfreferential. The paradox comes when you ask whether or not nonselfreferential is nonselfreferential. If it is, it doesn't refer to itself, so that nonselfreferential is not nonselfreferential. And if it isn't, nonselfreferential is selfrefential, so nonselfreferential is nonselfreferential. Contradiction!

Our friends in the arts, in literature, in philosophy, and in many aligned areas may be surprised to learn that this conflation plays a central role in mathematics and computer science.

2. Algorithms

Algorithms are the essence of computers. They are a precise set of instructions which take an *input* and produce an *output*. This broad view includes Turbo-Tax, input being a sequence of numbers and output being a completed tax return. Here, however, we will restrict our attention to a few specific possible types of inputs and outputs.

`Numerical Algorithms` Here the input is a positive integer, which we denote by n.

`Text Algorithms` Here the input is a text, generally denoted by **`TEXT`**.

Mixed Algorithms Here there are two inputs, a positive integer n and a text. This is rather specialized but plays a key role in selfreference.

While outputs in general can be of many forms we will restrict ourselves to *Boolean* responses.

YES The input has the desired property.

NO The input does not have the desired property.

Most importantly, however, there is a third possibility.

NONE No output is given. To avoid technical difficulties, we'll stipulate that if an algorithm is completed and no output has yet been given, then the output **NO** will be given. **NONE** occurs when the algorithm spins its wheels forever (often called an infinite loop) never reaching a conclusion. We further stipulate that **YES** and **NO** are the only possible outputs and that when the algorithm outputs one of these it then halts.

Here are some numerical algorithms.

Prime – PR Divide n by all the integers $2 \leq i \leq n - 1$. If none divide exactly then n is a prime. In that case the algorithm returns **YES**. If it ever does then n is not a prime. In that case the algorithm returns **NO**.

Sum of Two Squares – S2S Take all integers $1 \leq i \leq n$ and check whether $n - i^2$ is a square. If it ever is we have $n - i^2 = j^2$ so n is the sum of two squares. In that case the algorithm returns **YES**. If it never is then n is not the

sum of two squares. In that case the algorithm returns **NO**.

Difference of Two Squares — D2S Take all integers i and check whether $n + i^2$ is a square. If it ever is we have $n + i^2 = j^2$ so $n = j^2 - i^2$ is the difference of two squares. In that case the algorithm returns **YES**.

Algorithms **PR**, **S2S** are called *decision procedures*. They will definitely end and when they end we will know whether or not the input n has the property. **D2S** is different. If n is the difference of two squares then **D2S** will eventually find the expression of n as the difference of two squares and return **YES**. *But*, if n is not the difference of two squares **D2S** will go on forever, searching every higher i, never knowing for sure whether n is or is not the difference of two squares. (A subtle point here. We don't allow algorithms to go on forever and then look at what happened.)

Is there a *decision procedure* for difference of two squares. Yes!

Clever Difference of Two Squares — CD2S Apply **D2S**. However, when we reach i so big that $2i + 1 > n$ then stop the algorithm and return **NO**. In that case n is not the difference of two squares.

Why? Well, once i gets that big $i^2 + n$ lies strictly between i^2 and $(i + 1)^2 = i^2 + (2i + 1)$ and so it can't be a square. So we can stop — geeks like the word **HALT** — the procedure.

Let's be careful to distinguish the property from the algorithm. There

may be, as in **D2S**, **CD2S** many different algorithms which will say when the input *n* has the given property.

Text algorithms are equally important but simple examples are hard to come by. Here is one:

PALINDROME — PAL If the input **TEXT** is empty or contains a single letter then output **NO**. Otherwise, check the first and last letters of **TEXT**. If they are not the same output **NO**. If they are the same remove these letters from the text Otherwise remove those letters from **TEXT**. Then apply **PALINDROME**.

PAL is a decision procedure which outputs **YES** when **TEXT** is a palindrome and **NO** otherwise. It has a recursive form, calling itself. (Recursive algorithms come up frequently, those in the field will recognize **MERGESORT** as a typical example.) This particular algorithm will always end. Everytime the first and last letters are lopped off the **TEXT** becomes shorter by two so that eventually either **NO** is outputted or **TEXT** reaches length one or zero and **YES** is outputted.

3. Gematria and ASCII

Methods to translate texts into numbers range from the ancient Hebrew Gematria to the current ASCII code in computers. We want a two-way translation, where we can go both from the text to the number *and* from the number to the text. One way quite close to ASCII: Each letter (including blanks and digits

and other characters) *a, b,* … is assigned a two digit number 01,02, … These two digit numbers are then concatenated. For example, with blank being assigned 00,

"for example" becomes 06151800052401131612 05

(ASCII actually uses eight digits and is in binary.) It's important that each assignment is the same number of digits (in this approach, there are other methods such as *Human codes* that also work) as then the parsing is unique. So 0615180005240113161205 is parsed 06/15/18/00/05/24/01/13/16/12/05 which becomes "for example".

4. Self Reference

An algorithm, such as **PR** is itself a text. This is of course true, as anything that is written down is a text. But now algorithms have a double usage. In their "natural" setting they act on inputs to produce outputs. But simultaneously they, as texts, may be considered as inputs themselves to some other algorithm. Best of all, they may be inputs to themselves.

Mathematicians prefer to work with numbers so let us consider a numerical algorithm. Our gematria associates this text with a number n. Now we, critically, imagine the algorithm applied when n is the number of the algorithm itself. For example, apply **PR** with $n = 0409\cdots$, the number for "Divide…" There are three possibilities: **YES**, **NO** or **NONE**.

With **PR** either **YES** or **NO** will be returned, with **D2S YES** and **NONE** are the options and for other algorithms all three possibilities might occur.

Definition 1 Let us call a numerical algorithm selfaffirming if when applied to its number it returns **`YES`**. Otherwise (either **`NO`** or **`NONE`**) we call it nonselfaffirming. Further, we call a number *n* selfaffirming if its associated **`TEXT`** is a selfaffirming numerical algorithm and otherwise we call it nonself-affirming.

One technical point: For many integers n the associated **`TEXT`** is not an algorithm but simply gibberish, like **`JFJIESIJ93-DFDALIAD83`**. In that case *n* is nonselfaffirming.

5. Turing and Church

We've written algorithms in English but they may seem slippery. We want them to give explicit recipes. Something like "Return **`YES`** if *n* is beautiful" is not allowed. But just what is allowed? Fortunately, Alan Turing investigated this question in the 1930s. He created the now eponymous notion of a *Turing Machine*. This is a totally formal notion of algorithm. Our loosely written **`PR`**, **`S2S`**, **`D2S`**, **`CD2S`** can be expressed as Turing Machines. Does the notion of Turing Machine correspond to our human notion of an explicit recipe? This is our faith. Indeed it is called Church's Thesis. Kurt Gödel, Alan Turing, and the American mathematician Alonzo Church all approached this problem from somewhat different vantage points and I suspect Church's name is the most frequently used as the connection to religion is difficult to resist.

6. Harder Algorithms

Consider the property that n can be written $n = i^2 - 2j^3$. For example, 10 has this property as $10 = 8^2 - 2 \cdot 33$. Let's call such *n weird*. We can make an algorithm as follows:

SQUAREMINUSTWICECUBE — SM2C For each integer j check if n + 2j3 is a square. If it ever is, return **YES**.

If n is weird then **SM2C** will return **YES**. If n is not weird then **SM2C** will spin its wheels forever, never knowing for sure that n is not weird. Unlike **D2S**, I don't know of a decision procedure for weird. That **SM2C** is itself not a decision procedure for weird doesn't mean that there isn't a decision procedure, as the example **CD2S** illustrates. Indeed, while this author doesn't know of one, it's quite possible that those more knowledgeable of this area of number theory could give one.

Here is one that no one (today) knows. A *perfect* number, as defined by the ancient Greeks, is an n such that when you add up all of its divisors, excluding itself, you get back the original number. For example $28 = 1 + 2 + 4 + 7 + 14$ is perfect. Perfect numbers are quite rare, to date only about fifty of them have been found.

BIG ODD PERFECT — BOP Start checking, beginning at n and going up, for odd perfect numbers. When, and if, you find one return **YES**. Otherwise keep trying.

What does this do. If there are any odd perfect numbers n or higher then **BOP** returns **YES**. Otherwise it spins its wheels forever.

What happens to **BOP** when, say, $n = 3$. Well, if there are any odd perfect numbers it will find the first one and return **YES**. If there are no odd perfect numbers it will spin its wheels forever. As of today no one has found an odd perfect number and no one has proven that there are no odd perfect numbers. So, as of today, *we do not know* what will happen.

Of course, things could change tomorrow in either direction. Mathematicians are both searching for odd perfect numbers and searching for a *proof* that none exist. Perhaps they will succeed. Perhaps not.

7. Hilbert Hubris

David Hilbert was the preeminent mathematician of the early 20th century. His view was commonly held: all truths about the natural numbers could be demonstrated by thought, our thought. He wrote:

> One of the things that attracts us most when we apply ourselves to a mathematical problem is precisely that within us we always hear the call: here is the problem, search for the solution, you can find it by pure thought, for in mathematics there is no *ignorabimus*

His faith was shattered by the work of Kurt Gödel in 1931. Gödel showed, in a very fundamental way, that there would be statements about the positive integers that could neither be proven nor disproven. His arguments also used

the selfrefential notions — the felicitous confusion between text (for Gödel, proof) and number. Gödel's result was arguably the most important mathematical result of the 20th century. Certainly if one considers its philosophical impact there are no competitors. Over time the notions of unprovability and undecidability have been absorbed by the mathematical community. One can make analogies to the effects of Quantum Mechanics on Newtonian Physics. Yes, the change is absorbed, but the entire sense of the subject has been irrevocably altered.

8. Selfreference and Contradiction

How can we tell if a given numerical algorithm will return **YES** to a given input n. The remarks of section 6 show that this would be a tall order. Hilbert's faith (pre-Gödel) would tell us that human ingenuity would always find the answer. And Church's Thesis would tell us that this ingenuity could be turned into a recipe, an algorithm. Lets give this mythical algorithm a name.

Hilbert Church Investigator — HCI A mixed algorithm with a text input **TEXT** and a numerical input n. The algorithm will consider **TEXT** as a numerical algorithm and analyze what would happen if the input was that particular number n. It would return **YES** if the **TEXT** would have returned **YES**. Otherwise (whether the output were **NO** or **NONE** or if **TEXT** were gibberish) it would return **NO**.

Theorem HCI does not exist!

That is, not only have we not created **HCI** but we can never create **HCI** because no algorithm that purports to be **HCI** always gives the correct result. Algorithms can play world class chess and Go. They can drive cars and translate language. But *no* algorithm can determine whether or not a given numerical algorithm on a given input will output **YES**. We emphasize the insistence that the algorithm *always* give the correct answer.

The argument is a classical *reductio ad absurdum*. We imagine **HCI** does exist. We could then modify it to a numerical algorithm **SA** as follows. Take the number n, parse it and create the corresponding text **TEXT**. Consider **TEXT** as an numerical algorithm. Apply our mythical **HCI** to **TEXT** and n. (In Computer Science language, make **HCI** a subroutine.) If **HCI** would output **YES** then have **SA** output **YES**. Otherwise, whether **HCI** would yield **NO** or **NONE**, have **SA** output **NO**. The resulting numerical algorithm **SA** would output **YES** if and only if n is selfaffirming. One final tweak. Reverse the output, giving an algorithm **RSA**. That is, when **SA** is to output **YES**, have **RSA** output **NO** and when it is to output NO, have it output **YES**. In other words: **RSA** would output **YES** if and only if n is nonselfaffirming.

What's wrong with that? It leads to the classic selfreferential paradox of Section 1. **RSA** would itself be an algorithm, hence a text, hence associated to a number n. What would happen when **RSA** is given *its own number n*. We just said that **RSA** would output **YES** on input n if and only if n is nonselfaffirming. But n translates to **RSA** and so n is nonselfaffirming if and only if **RSA** would

not output **YES** on input n. We have reached a paradox. Something must be wrong. And what is wrong is our assumption that the mythical **HCI** could exist. Quod erat demonstrandum.

A great deal of quite technical work has gone into creating other scenarios in which undecidability appears. The most striking example is known as Hilbert's Tenth Problem — named for a sequence of problems Hilbert posed at the beginning of the last century. His problem generalized the numerical algorithms **S2S**, **D2S**, and **SM2C** previously discussed. Suppose, he asked, we are given some polynomial with integer coefficients in some number of variable. In the examples these were $i^2 + j^2$, $i^2 - j^2$, $i^2 - 2j^3$ respectively — but we make allow other variables k, l, ... Can we determine if an input integer n can be written in this form. For example, is there a solution to the equation

$$8419 = i^7 + j^8 - 83k^9$$

Critically, the variables (here i, j, k) must be *integers*. These are called *Diophantine* equations in the literature.

When Hilbert posed this problem he had no doubts that a solution, an algorithm in modern parlance, would be found. Gödel's 1931 bombshell gave another possibility — that it would be impossible to find a general solution. Still, Gödel's work created statements that were more complicated than polynomials. The quest to "solve" Hilbert's tenth problem was given new impetus but it proved to be technically quite challenging. The key step was found by

Martin Davis, Hilary Putnam and Julia Robinson in the mid 1960s. (Davis, now emeritus, was a colleague of this writer at the Courant Institute for many years.) The final step was made by Yuri Matiyasevich in 1970, proving that no such algorithm *can* exist.

9. Personal Reflections

We have not found a true paradox, we have not found a contradiction in the foundations of analytic thought. The argument was *reductio ad absurdam* and so the final conclusion was simply that the hypothesis was incorrect.

Still, one cannot underestimate the tectonic shift this has caused in the philosophy of mathematics and, in very practical terms, the way mathematicians like myself look at our subject.

I well recall as a high school student when I first learned of the twin prime conjecture. Twin primes are pairs of primes (such as 17, 19 or 101, 103) precisely two apart. The Twin Prime Conjecture is that there are an infinite number of them. I had an epiphany. This was an absolute question that demanded an absolute answer. I can imagine a world with silicon based life forms. I can imagine a universe with inverse cube gravitational forces. But I cannot imagine one universe in which there are an infinite number of primes and another universe in which there are only a finite number of primes. Mathematicians were searching for absolute truth.

There has been recent progress on the Twin Prime Conjecture. Yitang Zhang, in 2013, showed that were infinitely many pairs of primes within 70 million of each other, a number that has been whittled down to under 300. Possibly the full conjecture will be proven. We all hope so.

Paul Erdős (1913–1996) was a giant of 20th century mathematics. He was, is, and will be the most important person in my mathematical life, and this is true for countless others. Erdős spoke frequently of The Book. The Book contains all the theorems of mathematics and for each theorem it contains one proof. It is the best proof, the most insightful proof, the most beautiful proof — Paul called it *The Book Proof*. Our mission was to first search for a proof (or disproof) of the great conjectures. When someone succeeded and the proof was, well, ugly, Paul would say "This is very good. But now let's look for the Book Proof." This notion resonated deeply with all of us. The Book has a Platonic Reality. The proofs were already there. We did not create mathematics, we were to read the Pages of The Book. One time I gave an introduction for Paul to a group of talented high school students. I spoke about The Book but made the error of saying that it was held by God. Paul gave a gentle correction that I shall never forget. *You don't have to believe in God,* he said, *but you should believe in The Book.*

In my career I have worked on many problems and have had my share of moderate successes. But very many of the problems have resisted the strongest efforts of myself and my colleagues. Some problems get partial results but

others are brick walls. My favorite problem is the asymptotics of the Ramsey function $R(k,k)$. Results were found in 1931 and 1946, the latter by Erdős in the month of my birth. I have worked on this problem my entire career with little to show for it. Are we, simply, not smart enough to resolve the questions? Perhaps. But perhaps the problems fall into the deep well of undecidability. We continue to work but a voice in our heads will not be silent. It says: *Your quest may be in vain.*