arXiv:1409.5944v1 [math.LO] 21 Sep 2014

Gödel for Goldilocks: A Rigorous, Streamlined Proof of Gödel's First Incompleteness Theorem, Requiring Minimal Background[1]

Dan Gusfield
Department of Computer Science, UC Davis
August 29, 2014

1 Introduction: Why I wrote this

Gödel's famous incompleteness theorems (there are two of them) concern the ability of a formal system to state and derive all true statements, and only true statements, in some fixed domain; and concern the ability of logic to determine if a formal system has that property. They were developed in the early 1930s.

Most discussions of Gödel's theorems fall into one of two types: either they emphasize perceived philosophical "meanings" of the theorems, and maybe introduce some of the ideas of the proofs, usually relating Gödel's proofs to riddles and paradoxes, but do not attempt to present rigorous, complete proofs; or they do present rigorous proofs, but in the traditional style of mathematical logic, with all of its heavy notation and difficult definitions, and technical issues which reflect Gödel's original exposition and needed extensions by Gödel's contemporaries. Many people are frustrated by these two extreme types of expositions[2] and want a complete, rigorous proof that they can understand.

Over time, some people have realized that Gödel's first incompleteness theorem can be rigorously proved by a simpler middle approach, avoiding philosophical discussions and hand-waiving at one extreme; and also avoiding the heavy machinary of mathematical logic at the other extreme. This is the just-right *Goldilocks* approach. However, the available expositions of this middle approach have still been aimed at a relatively sophisticated audience, and have either been sketchy,[3] or have been

[1] Other than common things known to any moderately educated person, you only need to know what an integer is; what a function is; and what a computer program is. This lecture is intended to be given on the first day of my course offering "The Theory of Computation", which is a sophomore level course. You do need about two hours, and you need to focus — the material is concrete and understandable and requires almost no background, but it is not trivial.

[2] To verify this, just randomly search the web for questions about Gödel's theorem.

[3] For example, in Scott Aaronson's book "Quantum Computing Since Democritus".

embedded in larger, more involved discussions.[4] A short, self-contained Goldilocks exposition of Gödel's first theorem, aimed at a broader audience, has been lacking. Here we provide such an exposition.

We give a complete, rigorous proof that in *any* "rich-enough" formal proof system (defined below) that *never* derives a false statement, there are true statements that *cannot* be derived. That is a version of Gödel's first incompleteness theorem.

2 There are Non-Computable Functions

We will need to define several terms used here, and will do so at the appropriate times. We first discuss "computable" and "non-computable" functions.

Definition We use Q to denote all functions from the positive integers to $\{0, 1\}$. That is, if f is in Q, then for any positive integer x, $f(x)$ is either 0 or 1.

Definition Define a function f in Q as *computable* if there is a program (in Python, for example) that executes on a computer (a MacBook Pro running Snow Lepard, for example) that computes function f. That is, given *any* positive integer x, the program finishes in finite time and correctly spits out the value $f(x)$. Let A be the set of functions in Q that are computable.

Theorem 2.1 *There are functions in Q that are not computable; that is, $A \subset Q$.*

Proof Choose a computer language and consider all programs in that language that compute a function in A. Each line in a program has some end-of-line symbol, so we can concatenate the lines together into a single long string. Therefore, we think of a program as a single string written in some finite alphabet.

Next, conceptually, order the strings (representing the programs that compute the functions in A) into a list L, in order of the length of the strings, breaking ties arbitrarily. So each program that computes a function in A has a well-defined position in L. A function in A might be computed by different computer programs, so a function f in A might be represented in L more than once, but that will not matter. So L can be thought of as an ordered list of the functions in A, where a function can be listed more than once. Let f_i denote the function in A that is computed by the i'th program in L. (Remember that list L is only conceptual; we don't actually build it - we just have to imagine it for the sake of the proof).

Next, consider a table T with one colum for each positive integer, and one row for each program in L; and associate the function f_i with row i of T. Then set the

[4] For example, Sipser's classic and excellent book on the Theory of Computation, where the exposition of Gödel's theorem relies on an understanding of Turing machines and the Undecidability of the Halting problem.

value of cell $T(i, x)$ to $f_i(x)$. See Figure 1. Since L is only conceptual, T is also only conceptual.

Now define function $\overline{f}$ from the positive integers to $\{0, 1\}$ as $\overline{f}(i) = 1 - f_i(i)$. For example, based on the functions in Figure 1, $\overline{f}(1) = 0; \overline{f}(2) = 1; \overline{f}(3) = 1; \overline{f}(4) = 0; \overline{f}(5) = 1$

Note that in the definition of $\overline{f}(i)$, the same integer i is used both to identify the function f_i in A, and as the input value to f_i and to $\overline{f}$. Hence the values for $\overline{f}$ are determined from the values along the main *diagonal* of table T. Note also that $\overline{f}$ changes 0 to 1, and changes 1 to 0. So, the values of function $\overline{f}$ are the *opposite* of the values along the main diagonal of Table T. Clearly, function $\overline{f}$ is in Q.

	1	2	3	4	5	.	x	.	i	...
f_1	1	1	0	0	0		$f_1(x)$			
f_2	0	0	1	0	0		$f_2(x)$			
f_3	1	1	0	0	1		$f_3(x)$			
f_4	0	0	1	1	0		$f_4(x)$			
f_5	0	1	0	0	0		$f_5(x)$			
.						.				
.							.			
.								.		
f_i									$f_i(i)$	
⋮										⋱

Table 1: The conceptual Table T enumerates all computable functions (from positive integers to $\{0, 1\}$), and their values at all of the integers.

Now we ask: Is $\overline{f}$ a computable function? The answer is no for the following reason. If $\overline{f}$ were a computable function, then there would be some row i^* in T such that $\overline{f}(x) = f_{i^*}(x)$ for any positive integer x. For example, maybe i^* is 57. But $\overline{f}(57) = 1 - f_{57}(57) \neq f_{57}(57)$, so $\overline{f}$ can't be f_{57}. More generally, $\overline{f}(i^*) = 1 - f_{i^*}(i^*)$, so $\overline{f}$ and f_{i^*} differ at least for one input value (namely i^*), so $\overline{f} \neq f_{i^*}$. Hence, there is no row in T corresponding to $\overline{f}$, and so $\overline{f}$ is not in set A. So $\overline{f}$ is not computable — it is in Q, but not in A. ■

3 What is a Formal Proof System?

How do we connect Theorem 2.1, which is about functions, to Gödel's first incompleteness theorem, which is about logical systems? We first must define a *formal proof system*.

Definition A formal proof system Π has three components: 1) A finite alphabet, and some finite subset words and phrases that can be used in forming (or writing) *statements*.[5] 2) a finite list of *axioms* (statements that we take as true); and a finite list of *logic rules*, also called *deduction or derivation* rules, that can be applied to transform, in a precise mechanical way, one statement into another.

For example, the alphabet might be the standard Ascii alphabet with 256 symbols. A first axiom might be the statement "for any integer x, $x+1 > x$." A second axiom might be "for any integers x and y, $x+y$ is an integer." A logic rule might be "for any three integers, x, y, z, if $x > y$ and $y > z$ then $x > z$." (Call this rule the "transitivity rule".) Another rule might be "for any two statements represented by p and q, if p and q always have the same value then you can replace p with q in any statement containing p." (Call this the "Replacement rule".) The finite set of English words and phrases might include the phrase "for any integer". Of course, there will typically be more axioms, logic rules and known words and phrases than in this example.

3.1 What is a Formal Derivation?

Definition A *formal derivation* in Π of a statement S is a series of statements that begin with some axioms of Π, and then successively use some of the logic rules in Π to get to statement S.

For example, S might be the statement: "for any integer x, $x+1+1 > x$". A formal derivation of S in Π might be:

- For any integer x, $x+1 > x$ (axiom 1).
- x is an integer, 1 is an integer, so $x+1$ is an integer (by axiom 2).
- Define y to be equal to $x+1$ and use the replacement rule on the first statement, resulting in: $y > x$.
- y is an integer since $x+1$ is an integer (more formally, this comes from replacing $x+1$ by y in the second part of the second statement above).
- $y+1 > y$. (by axiom 1, since y is an integer).
- Using the replacement rule, replace y with $x+1$, resulting in: $x+1+1 > x+1$.
- Use axiom 1 (i.e., $x+1 > x$) and the transitivity rule to conclude that $x+1+1 > x$, which is statement S.

[5] These words and phrases are strings in the alphabet of Π. We will say that they are a subset of English, but it would be historically more correct to say they are from German.

The finite subset of English used in this formal derivation includes the words and phrases "Define", "to be equal", "Using the replacement rule" "We conclude", etc. These would be part of the finite subset of English that is part of the definition of Π. Each phrase used must have a clear and precise meaning in Π, so that each non-axiom statement in a formal derivation follows in a mechanical way from the preceeding statements by the application of some logic rules. Formal derivations are very tedious, and we as humans don't want to write proofs or derivations this way, but computers can write and check them, as we will explain below. (Note that what I have called a "formal derivation" is more often called a "formal proof". But that is confusing, because people usually think of a "proof" as something that establishes a *true* statement, not a statement that might be false. So here we use "formal derivation" to avoid that confusion.)

3.2 Mechanical Generation and Checking of Formal Derivations

We now make three key points about formal derivations.

1. The first key point is that a formal derivation, being a series of statements, is just a string formed from the alphabet and the allowed words and phrases of the formal proof system Π. Hence, it is easy to write a program P that can begin generating, in order of the length of the string, every string s that can be written in the formal proof system Π. Program P will never stop because there is no bound on the length of the strings, but for any finite-length string s using the alphabet of Π, P will eventually (and in finite time) generate s.

2. The second key point about a formal proof system is that we can create a program P' that knows the alphabet, the axioms, the deduction rules, and the meaning of the words of the subset of English used in Π, so that P can precisely interpret the effect of each line of a formal derivation. That is, P can mechanically check whether each line is an axiom, or follows from the previous lines by an application of some deduction rule(s). Therefore, given a statement S, and a string s that might be a formal derivation of S, program P' can check (in a purely mechanical way, and in finite time) whether string s is a formal derivation of statement S in Π.

3. The third key point is that for any statement S, after program P generates a string s, program P' can check whether s is a formal derivation of statement S in Π, before P generates the next string. Hence, if there is a formal derivation s in Π of a statement S, then s will be generated and recognized in finite time by interleaving the execution of programs P and P'.

Note that most of the strings that P generates will be garbage, and most of the strings that are not garbage will not be formal derivations of S in Π. But if string

s is a formal derivation of statement S, then in finite time, program P will generate s and program P' will recognize that s is a formal derivation in Π of statement S. Similarly, we can have another program P'' which checks whether a string s is a formal derivation the statement "not S", written $\neg S$. So if $\neg S$ is a statement that can be derived in Π, the interleaved execution of programs P and P'' will, in finite time, generate and recognize that s is a formal derivation of $\neg S$.

4 Back to Gödel

Now how do we connect all this to Gödel's first incompleteness theorem? We want to show that in any "rich-enough" formal proof system Π that only derives true statements, there are true statements that can not be derived in Π. We haven't defined what "true" or "rich-enough" means in general, but we will in a specific context.

Recall function $\overline{f}$, and recall that it is well-defined, i.e., there is a value $\overline{f}(x)$ for every positive integer x, and for any specific x, $\overline{f}(x)$ is either 0 or 1. Recall also, that $\overline{f}$ is not a computable function.

Definition We call a statement an $\overline{f}$-statement if it is either:

"$\overline{f}(x)$ `is` 1,"

or:

"$\overline{f}(x)$ `is` 0,"

for some positive integer x.

Note that every $\overline{f}$-statement is a statement about a specific integer. For example the statement "$\overline{f}(57)$ `is` 1" is an $\overline{f}$-statement, where x has the value 57. Since, for any positive integer x, $\overline{f}(x)$ has only two possible values, 0 or 1, when the two kinds of $\overline{f}$-statements refer to the same x, we refer to the first statement as $Sf(x)$ and the second statement as $\neg Sf(x)$.

What is Truth? We say an $\overline{f}$-statement $Sf(x)$ is "true", and $\neg Sf(x)$ is "false", if in fact $\overline{f}(x)$ is 1. Similarly, we say an $\overline{f}$-statement $\neg Sf(x)$ is true, and $Sf(x)$ is false, if in fact $\overline{f}(x)$ is 0. Clearly, for any positive integer x, one of the statements $\{Sf(x), \neg Sf(x)\}$ is true and the other is false. In this context, truth and falsity are simple concepts (not so simple in general). Clearly, it is a natural and desirable property of a formal proof system Π, that it is not possible to give a formal derivation in Π for a statement that is false.

What does it mean to be rich-enough? We need a definition.

Definition We define a formal proof system Π to be *rich-enough* if any $\overline{f}$-statement can be formed (i.e., stated, or written) in Π.

Note that the words "formed", "stated", "written" do not mean "derived". The question of whether a statement can be derived in Π is at the heart of Gödel's theorem. Here, we are only saying that the statement can be formed (or written) in Π. For example, a formal proof system that can form *any* statement about integers is rich-enough, since $\overline{f}$ is a function from a subset of integers to subset of integers.

4.1 The Proof of Gödel's Theorem

Now suppose **a)** there is a rich-enough formal proof system Π; and suppose **b)** that Π has the properties that only true statements can be derived in Π, and suppose **c)** that for any true statement S that can be formed in Π, there is a formal derivation of S in Π.

Since Π is rich-enough, for any positive integer x, both statements $Sf(x)$ and $\neg Sf(x)$ can be formed in Π, and since exactly one of those statements is true, there will be a formal derivation in Π of exactly one of the two statements, in particular, the statement that is true. But this leads to a contradiction of the established fact that function $\overline{f}$ is not computable.

In more detail, if the three suppositions hold, the following approach describes a computer program that can correctly determine, in finite time, the value of $\overline{f}(x)$, for any positive integer x:

> Given x, run program P to successively generate the strings in order of their lengths (using the finite alphabet and finite known words and phrases in Π), and after each string s is generated, run program P' to see if s is a formal derivation of statement $Sf(x)$. If it is, Eureka, and if it isn't, run P'' to see if s is a formal derivation of $\neg Sf(x)$. If it is, Eureka, and if it isn't, let P go on to the next string.

The three suppositions guarantee that for any positive integer x, this mechanical computer program will have a Eureka moment in finite time, revealing the correct value of $\overline{f}(x)$.

But then $\overline{f}$ would be a *computable* function, contradicting the already established fact that $\overline{f}$ is not a computable function. Hence, the three suppositions (a, b, c) cannot all hold for any rich-enough Π. There are several equivalent conclusions that result. One is that if a rich-enough formal proof system Π has the property that only true statements can be derived in Π, then there must be true statements that can be formed in Π but not derived in Π. That is, Π is *incomplete*.

4.2 Recapping

We have established:

Theorem 4.1 *For any rich-enough formal proof system* Π *in which no formal derivation of a false statement is possible, there will be some true* $\overline{f}$*-statement that cannot be formally derived in* Π*.*

So, for example, *any* formal proof system that can form any statement about integers, and only derive true statements, will be incomplete. That is (a version of) Gödel's first incompleteness theorem.

5 An Alternate Statement of Gödel's First Incompleteness Theorem

Gödel's first incompleteness theorem is often stated differently than the way it is stated above. To describe the variant of the theorem, we need:

Definition A formal proof system Π is called *consistent* if it is never possible to derive a statement S in Π and also derive the statement $\neg S$ in Π. Otherwise, Π is called *inconsistent*. A formal proof system Π is called *complete* if it is possible to derive in Π any true statement that can be formed in Π.

Gödel's first theorem can then be stated as:

Theorem 5.1 *No formal proof system* Π *that can form any statement about integers is both consistent and complete.*

Theorem 4.1 implies Theorem 5.1; we leave that to the reader to establish.

6 Gödel's Second Incompleteness Theorem

Later in the course, we will talk about Gödel's second incompleteness theorem, which requires only a small technical addition, but it needs more machinery. Informally, it says that if Π is rich-enough and consistent, there cannot be a formal derivation in Π of the statement: "Π is consistent". More philosophically, but not precisely, for any (rich enough) formal proof system Π that is consistent, the consistency of Π can only be established by a different formal proof system Π'. (But then, what establishes the consistency of Π'?)

7 Homework questions:

1. What is the point of requiring program P to generate strings in order of their lengths? Would the given proof of Gödel's theorem work if P did not generate the strings in order of length, but could (somehow) generate all the strings, but in no predictable order?

2. Doesn't the following approach show that $\overline{f}(x)$ is computable?

First, create a computer program P' that can look at a string s over the finite alphabet used for computer programs (in some fixed computer language, for example, C), and determine if s is a legal computer program that computes a function f in Q. Certainly, a compiler for C can check if s is a syntactically correct program in C.

Then given any positive integer x, use program P to generate the strings over the finite alphabet used for computer programs, in order of their length, and in the same order as used in table T. After each string s is generated, use program P' to determine if s is a program that computes a function in Q. Continue doing this until x such programs have been found. In terms of table T, that program, call it F, will compute function f_x. Program F has finite length, so P will only generate a finite number of strings before F is generated. Then once F is generated, run it with input x. By definition, program F will compute $f_x(x)$ in finite time. Then output $\overline{f}(x) = 1 - f_x(x)$.

So, this approach seems to be able to compute $\overline{f}(x)$ in finite time, for any positive integer s, showing that $\overline{f}$ is a computable function. Doesn't it?

Discuss and resolve.

3. Use the resolution to the issue in problem 2, to state and prove an interesting theorem about computer programs (yes, this is a vague question, but the kind that real researchers face daily).

4. Show that Theorem 4.1 implies Theorem 5.1. What about the other direction?

8 What is not in this exposition?

Lots of stuff that you might see in other proofs and expositions of Gödel's first incompleteness theorem: propositional and predicate logic, models, WFFs, prime numbers, prime factorization theorem, Gödel numbering, countable and uncountable infinity, self-reference, recursion, paradoxes, liars, barbers, librarians, "This statement is false", Peano postulates, Zermelo-Fraenkel set theory, Hilbert, Russell, Turing, universal Turing machines, the halting problem, undecidability, ..., quantum theory, insanity, neuroscience, the mind, zen, self-consciousness, evolution, relativity, philosophy, religion, God, Stalin, ... The first group of topics are actually related in precise,

technical ways to the theorem, but can be avoided, as done in this exposition.[6] Some of those related technical topics are important in their own right, particularly Turing undecidability, which we will cover in detail later in the course. The second group of topics are not related in a precise, technical way to the theorem. Some are fascinating in their own right, but their inclusion makes Gödel's theorem more mystical, and should not be confused for its actual technical content.

9 Final Comments

First, the exposition here does not follow Gödel's original proof, and while the exposition is my own, the general approach reflects (more and less) the contemporary computer-sciencey way that Gödel's theorem is thought about. In coming to this exposition for undergraduates, I must acknowledge the discussion of Gödel's theorems in Scott Aaronson's book *Quantum Computing Since Democritus*, and an exposition shown to me by Christos Papadimitriou. Those are both shorter, aimed at a more advanced audience, and are based on the undecidability of Turing's Halting problem.

Second, I would be (somewhat) dishonest if I didn't admit that the version of Gödel's theorem proved here is weaker than what Gödel originally proved. In this exposition, the formal proof system must be able to express any statement about integers (or at least any $\overline{f}$-statement) but Gödel's original proof only requires that the formal system be able to express statements about arithmetic (in fact, statements about arithmetic on integers only using addition and multiplication), which is a more limited domain, implying a stronger theorem. That difference partly explains why Gödel's original proof is technically more demanding than the exposition here. Further, Gödel did not just proove the *existance* of a true statement that could not be derived (in any rich-enough consistent proof system), he demonstrated a particular statement with that property. But, I believe that the moral, philosophical (oops, I used that word) impact of the what is proved here is the same as Gödel's original first incompleteness theorem, and most current treatments of Gödel's theorem similarly follow this view.

[6] Most expositions of Gödel's theorems use self-reference, which I find unnecessarily head-spinning, and I think its use is sometimes intended to make Gödel's theorem seem deeper and more mystical than it already is.