Gödel's Incompleteness Theorems

Kenny Easwaran

January 31, 2023

The goal of this handout is to cover the technical details of a proof of the Incompleteness phenomenon that Gödel identified. It assumes a familiarity with some sort of system of formal derivations to represent proofs in a formal language, though there is some review of the formal language for arithmetic. Along the way we will define a formal notion of "definability" for various properties of numbers, and show how to represent sentences of our formal language with numbers. The Incompleteness Theorem is then proved by showing that the set of code numbers of sentences derivable from the standard axioms of arithmetic is definable, while the set of code numbers of true sentences in the formal language of arithmetic is not definable. Thus, there must either be a derivable sentence that isn't true (which we can avoid by choosing a good set of axioms) or a true sentence that isn't derivable.

For some purposes, all we care about might be the fact that there are true sentences that aren't derivable from the standard axioms. However, the result we prove is easily generalized to a range of sets of axioms. We will argue that *any* good axiom system must be in this range, so that the problem can't be fixed.

Furthermore, the particular notion of definability we use has some very important features. We will argue that any set that a computer (or a human) could reasonably work with will be definable in our sense, which will show that the full concept of truth in arithmetic is actually much too complex for us to properly grasp.

Additionally, the result will apply to *any* field of study that is powerful enough to incorporate arithmetic.

1 The Formal Language for Arithmetic

The alphabet consists of the following symbols:

- Punctuation: $), ($
- Connectives: $\neg, \wedge, \vee, \rightarrow$
- Quantifiers: $\exists, \forall$
- Relations: $<, =$

- One-Place Function: S
- Two-Place Functions: $+, \cdot, E$
- Constant: 0
- Variables: $x, y, z, \ldots$

Terms are defined as follows:

1. Constants and variables are terms
2. If t is a term, then St is a term
3. If t and u are terms, then $(t + u), (t \cdot u), (tEu)$ are terms
4. Nothing else is a term*

Well-formed formulas (or *wffs*) are defined as follows:

1. If t and u are terms, then $(t < u)$ and $(t = u)$ are wffs (called *atomic* wffs)
2. If ϕ and ψ are wffs, then so are $\neg\phi, (\phi \wedge \psi), (\phi \vee \psi), (\phi \rightarrow \psi)$
3. If ϕ is a wff and v is a variable, then $\exists v\phi$ and $\forall v\phi$ are wffs
4. Nothing else is a wff*

Instances of variables are said to be *free* or *bound* depending on whether or not they're in the scope of a quantifier of the appropriate variable. A formula with no free variables is called a *sentence*. Sentences are the only things that are true or false.

*The definitions of terms and wffs are inductive definitions. The way to cash them out more rigorously is as follows. We say that a set is "good" if it contains all the objects listed under condition 1, and is closed under all the operations in conditions 2 and 3. We then say that something is a term (or wff) iff it is in every "good" set. This lets us use induction to prove things about every term (or wff) — if we prove that the set of objects having some property contains all the objects in condition 1, and that it is closed under the operations of 2 and 3, then this automatically proves that every term (or wff) has that property.

Exercise 1 Which of the following are terms?

x	$x + y$	$(x + (z \cdot y))$	$(S(z \cdot z) + (xE0))$
$Sx + Sy + S0$	$xE(yEz)$	$Sx \cdot 0$	$S(xE0)$

Exercise 2 Which of the following are wffs?

$\exists xS(0 = x)$	$S0 = x$	$((x < y) \vee (y < x))$	$(S(x + y) = (x + Sy))$
$\forall x\exists y(x < y)$	$\exists x(0 < 0)$	$\forall x\exists y(x < y \wedge \neg\exists x\exists z(y = (SSx \cdot SSz)))$	$\exists y\forall y\forall z\neg(x = x)$

Exercise 3 Which of the following wffs are sentences?

$\exists x\forall y(x < y)$	$(\forall x(x = 0) \vee (0 < x))$	$\forall y(x = x)$
$\exists x((x + SS0) = x)$	$(x < (y + y))$	$(S0 < 0)$

Truth

For this particular language, we will consider only one specific model and interpretation. In general, a logical language may have many interpretations (an argument is said to be *logically valid* iff there is no interpretation where the premises are true and the conclusion is false), but here we have a specific meaning in mind.

The domain here is the natural numbers (including zero). The constant 0 represents the number zero. The function S takes a number and returns its successor. The functions $+, \cdot, E$ represent addition, multiplication, and exponentiation respectively (we stipulate here that anything, including zero, raised to the zeroth power is one). The relations $<, =$ have their usual meanings of "less than" and "equal to".

This gives us a notion of truth for the atomic wffs. For the more complex wffs, we define truth in a recursive way, following the inductive definition. For the connectives, we just use the usual truth-tables:

A	B	$\neg A$	$(A \wedge B)$	$(A \vee B)$	$(A \rightarrow B)$
T	T	F	T	T	T
T	F	F	F	T	F
F	T	T	F	T	T
F	F	T	F	F	T

For the quantifiers it is slightly more complicated. Basically, the sentence $\exists v\phi$ is true iff there is some numeral n (a term of the form $SSSS\ldots0$) such that ϕ is true when every free instance of v is replaced by n. The sentence $\forall v\phi$ is true iff this holds for *every* numeral. For sentences with multiple nested quantifiers, this requires a recursive definition that tracks the recursive structure of the sentence. (Note that this sort of definition of truth is available only because we are working in a domain where every object has a name in the language. For more general interpretations, we need to work with a notion of "variable assignments" and define "satisfaction" before defining truth, instead of the reverse as we are about to do.)

Exercise 4 Which of the following sentences are true:

$((S0 + SS0) = SSS0)$

$\exists y\exists x((x + Sx) = SS(y + y))$

$\exists x((SS0 < x) \wedge ((xESS0) = (SS0Ex)))$

$\exists x\forall z((z < x) \rightarrow ((z = S0) \vee \neg\exists y((y \cdot z) = x)))$

$\exists x\exists y\exists z\exists n(((SxESSn) + (SyESSn)) = (SzESSn))$

$\forall x\exists y((x < y) \wedge \neg\exists z\exists w(y = (SSz \cdot SSw)))$

Satisfaction and Definability

If ϕ is a sentence, then it is either true or false, but there is a similar notion we can define for wffs in general. If ϕ is a wff with n free variables, then we say that terms $t_1, \ldots t_n$ *satisfy* ϕ iff the sentence $\phi[t_1, \ldots, t_n]$ obtained by substituting the kth free variable in ϕ with t_k is true.

We say that a property $X(x)$ of numbers is *definable* iff there is a wff ϕ with exactly one free variable such that the numeral for x satisfies ϕ iff $X(x)$. Similarly, for any n-place relation $R(x_1, \ldots, x_n)$, it is definable iff there is some wff $\phi(x_1, \ldots, x_n)$ satisfied exactly when this relation holds.

Exercise 5 Write down formulas showing that each of the following properties or relations is definable. Feel free to use earlier properties or relations in the definition of later ones, instead of writing out the entire formula in the official formal language. It is best to think of each of these predicates written with small caps as an abbreviation for some formula in the language. If you don't use these abbreviations, the formulas likely will get too long to fit on one line after the first few of them, and might even be too long to fit on the page by the end of this list (certainly by the end of the handout).

- $\textsc{Divides}(x, y)$ saying that y is divisible by x.
- $\textsc{Prime}(x)$ saying that x is prime (i.e., that x is not divisible by any number other than 1 or itself).
- $\textsc{NextPrime}(p, q)$ saying that p is prime and q is prime and there are no primes between them.
- $\textsc{DividesK}(p, n, k)$ saying that p divides n exactly k times (that is, p^k divides n but p^{k+1} does not).
- $\textsc{OneMore}(p, q, n)$ saying that p and q are both prime, and that q divides n exactly one more time than p does.
- $\textsc{PrimeSeq}(x, p)$ saying that p is prime, that x is divisible by 2 but not by 4, and that each prime up to p divides x exactly one more time than the prime before it.
- $\textsc{NthPrime}(p, n)$ saying that p is the nth prime. That is, $\textsc{NthPrime}(2, 1)$, $\textsc{NthPrime}(3, 2)$, $\textsc{NthPrime}(5, 3)$ should all hold.
- $\textsc{PPower}(x, n, k)$ saying that the nth prime divides x exactly k times. (That is, $\textsc{PPower}(45, 1, 0)$, $\textsc{PPower}(45, 2, 2)$, and $\textsc{PPower}(45, 3, 1)$ should all hold, but no other $\textsc{PPower}(45, x, y)$ where $y > 0$.)

2 Recap of Derivations, Soundness, and Completeness

A derivation system is a set of purely syntactic rules that can be used to generate a conclusion ϕ from a set of premises, Γ. We state that a derivation exists by saying "$\Gamma \vdash \phi$". We have so far only given the details of one semantic interpretation of our formal language, but the concept of logical validity for an argument states that there is no interpretation that makes every premise of the argument true while making the conclusion false. We state that the argument from Γ to ϕ is *valid* by saying "$\Gamma \models \phi$".

The Soundness Theorem of first-order logic states that if $\Gamma \vdash \phi$ then $\Gamma \models \phi$. The Completeness Theorem is the converse — if $\Gamma \models \phi$ then $\Gamma \vdash \phi$.

Recall that Soundness was proved directly by showing that every axiom in the derivation system is true on all interpretations, and every derivation

rule preserves truth, so that any interpretation making all the premises of the derivation true must make the conclusion of the derivation true. Recall that Completeness was proved by first translating it to the notions of consistency and satisfiability. We say that a set Γ of sentences is *inconsistent* iff there is some derivation of $\perp$ from Γ (equivalently, if there is some sentence ϕ such that $\Gamma \vdash \phi$ and $\Gamma \vdash \neg\phi$). We say that a set Γ of sentences is *satisfiable* iff there is some interpretation that makes every sentence of Γ true. We can show that $\Gamma \vdash \phi$ iff $\{\Gamma, \neg\phi\}$ is inconsistent, and that $\Gamma \models \phi$ iff $\{\Gamma, \neg\phi\}$ is unsatisfiable. Thus, Completeness is equivalent to the statement that every consistent set of sentences is satisfiable. We then prove this by extending the consistent set to a maximal consistent set with witnesses, and showing that a maximal consistent set with witnesses can be used to generate an interpretation where every one of its sentences is true.

3 Gödel Numbering

In order to prove Gödel's Incompleteness Theorems, it is necessary to associate sentences with numbers in order to associate properties of sentences with properties of numbers. We will then show that certain of these properties of numbers are definable and certain others are not, which proves that they must be distinct sets. Thus, there must be some number with one property but not the other. Since these properties are associated with properties of corresponding sentences, this means there must be a sentence with one property but not the other.

In particular, we will show that the set of numbers corresponding to sentences derivable from the Peano axioms (to be defined later) is definable, but the set of numbers corresponding to sentences that are true in the standard interpretation is not. Thus, there must either be a sentence derivable from the Peano axioms that is not true, or a true sentence that is not derivable from the Peano axioms. Since the Peano axioms are all clearly true, this means that the latter must hold — there must be a true sentence that is not derivable from the Peano axioms. Furthermore, the method of proof can be extended to a very wide range of axiom systems for number theory — we will show that *no* reasonable system of axioms can derive all and only the true sentences.

One way to associate numbers with sentences would be to enumerate all the sentences from shortest to longest, and then number them in some sort of "alphabetical order" within the lengths. But this numbering (and most others) will make it hard to show that the set of numbers associated with derivable sentences is definable. So instead we will associate numbers with each *symbol* in the language and then build the number for a sentence (or any other sequence of symbols) out of the numbers for its symbols.

I will arbitrarily fix this particular association of positive natural numbers to the symbols in the language:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...
)	(	$\neg$	$\wedge$	$\vee$	$\rightarrow$	$\exists$	$\forall$	$<$	$=$	S	$+$	$\cdot$	E	0	x	y	z	...

We then say that the Gödel number of a string of n symbols is generated by

taking the first n primes, raising each to the power of the code number of the corresponding symbol in the string, and multiplying them together. This coding system will not be very tractable or convenient for using — even the shortest possible sentence has a code number that is almost 30 digits long! ("$(0 < 0)$" is the shortest possible sentence, and its code number is $2^2 3^{15} 5^9 7^{15} 11^1 = 4 \cdot 14,348,907 \cdot 1,953,125 \cdot 4,747,561,509,943 \cdot 11$.) In practice, we will not deal with these actual numbers, but will just show that the appropriate sets and relations of them are definable.

The following exercises begin this by showing that certain relations among sequences correspond to definable properties of the code numbers:

Exercise 6 Write down formulas showing that the following properties or relations are all definable (you'll need to use some sets and relations that you defined in the previous set of exercises):

- $\textsc{String}(x)$ saying that x is the code number of a string of symbols. (Note that every symbol has a code number greater than 0, and so a string of symbols will always have a code number that is divisible by several consecutive early primes, and then no later primes.)
- $\textsc{Length}(x, k)$ saying that x is the code number of a string of exactly k symbols.
- $\textsc{StringConcat}(x, y, z)$ saying that x, y, z are all codes of strings of symbols, and that x is the code of the string achieved by concatenating y's string with z's string. (This one is somewhat long and a bit tricky.)
- $\textsc{Substring}(x, y)$ saying that x is the code for a substring of the string coded by y. (That is, the symbols of x's string appear somewhere consecutively and in the right order in y's string, though there may be extra symbols earlier or later in y's string.)

The next exercises show that some particular syntactic relations among sequences of symbols correspond to definable properties of the code numbers:

Exercise 7 Write down formulas showing that the following properties or relations are all definable (you'll need to use some sets and relations that you defined in the previous set of exercises):

- $\textsc{Numeral}(x, y)$ saying that x is the code number of the numeral for y (that is, a string of exactly y S's followed by a 0).
- $\textsc{Succ}(x, y)$ saying that x, y are the codes of strings of symbols, and that x's string is just "S" followed by y's string.
- $\textsc{Neg}(x, y)$, saying that x, y are the codes of strings of symbols, and that x's string is just "$\neg$" followed by y's string.
- $\textsc{Universal}(x, y, z)$, saying that x, y are the codes of strings of symbols, that z is the code of a variable v, and that x's string is just "$\forall v$" followed by y's string.

- EXIST(x, y, z), saying that x, y are the codes of strings of symbols, that z is the code of a variable v, and that x's string is just "$\exists v$" followed by y's string.
- PLUS(x, y, z) saying that y is the code of some string ϕ, z is the code of some string ψ, and x is the code of the string $(\phi + \psi)$.
- TIMES(x, y, z) saying that y is the code of some string ϕ, z is the code of some string ψ, and x is the code of the string $(\phi \cdot \psi)$.
- EXP(x, y, z) saying that y is the code of some string ϕ, z is the code of some string ψ, and x is the code of the string $(\phi E \psi)$.
- LESS(x, y, z) saying that y is the code of some string ϕ, z is the code of some string ψ, and x is the code of the string $(\phi < \psi)$.
- EQUAL(x, y, z) saying that y is the code of some string ϕ, z is the code of some string ψ, and x is the code of the string $(\phi = \psi)$.
- CONJ(x, y, z) saying that y is the code of some string ϕ, z is the code of some string ψ, and x is the code of the string $(\phi \wedge \psi)$.
- DISJ(x, y, z) saying that y is the code of some string ϕ, z is the code of some string ψ, and x is the code of the string $(\phi \vee \psi)$.
- IFTHEN(x, y, z) saying that y is the code of some string ϕ, z is the code of some string ψ, and x is the code of the string $(\phi \to \psi)$.

Sequences of Strings

At this point we can't yet give formulas defining TERM(x) and WFF(x) saying that x is the code number of a term or wff. Because those definitions are given inductively, we need to consider sequences of strings, instead of just single strings — after all, whether or not something counts as a term (or wff) depends on whether the things it's built out of do.

We can assign a code number to a sequence of strings of symbols by doing the same trick, of raising consecutive primes to the power of each one. The numbers we get this way are of course going to be astronomical, but fortunately we won't deal with them by hand - we'll just write down formulas defining things about them.

Exercise 8 Write down formulas showing that the following sets or relations are all definable:

- SEQ(x) saying that x is the code of a sequence of strings.
- LAST(x, y) saying that y is the code of a sequence of strings, and x is the code of the last string in y's sequence.

- TermFormSeq(x) saying that x is the code of a term formation sequence. That is, a sequence of strings, such that each string is either a variable or constant, or is the Succ of some previous string in the sequence, or the Plus or Times or Exp of two previous strings.
- Term(x) saying that x is the code of a term.
- Atomic(x) saying that x is the code of an atomic wff.
- WffFormSeq(x) saying that x is the code of a wff formation sequence. That is, a sequence of strings, such that each string is either an atomic wff, or is the Neg of some previous string in the sequence, or the Conj or Disj or IfThen of two previous strings, or the Universal or Exist of a previous string with some variable.
- Wff(x) saying that x is the code of a wff.

Once we have defined Wff(x), we can also define Free(x, k) and Bound(x, k), saying that x is the code of a wff, and the kth symbol in x is a variable, which is respectively free or bound. The way we do this is basically to say that the kth symbol in x is a variable, and occurs (or doesn't occur) in some substring of x's string which is itself a wff, and which begins with a quantifier on the variable in the kth place.

Satisfaction and Truth

So far we've shown that a bunch of things are definable — is there anything that isn't definable? The following observations (in a modified form) are due to Alfred Tarski.

Let the relation Substit(x, y, z) hold iff x is the code number of a sentence ϕ, y is the code number of a wff ψ with one free variable, and z is the code number of a term t, and ϕ is $\psi[t]$. This relation is definable (continuing the sort of work we've done above will be able to show this).

Let Sat(y, z) hold iff y is the code of a wff with exactly one free variable, satisfied by a term consisting of z S's followed by a 0.

If True(x) were definable, then we could define Sat(y, z) as follows:

$$\exists x \exists z' (\text{Substit}(x, y, z') \wedge (\text{Numeral}(z', z) \wedge \text{True}(x)))$$

Thus, if we can show that Sat(y, z) is undefinable, then we could show that True(x) was undefinable.

If Sat were definable, then consider the formula $\neg$Sat(x, x), and let its code number be r.

Now consider whether $\neg$Sat($S \ldots S0, S \ldots S0$) is true, where both terms are the numeral for r.

It is true iff r is the code of a wff with exactly one free variable (which it is, as we can see), which is not satisfied by a term consisting of r S's followed by

a 0. But this just means asking whether $\neg\text{SAT}(x, x)$ is not satisfied by r. But that is the case iff $\neg\text{SAT}(S \ldots S0, S \ldots S0)$ is false.

Thus we have a contradiction, which means that our assumption was false. Since our only assumption was that $\text{SAT}(x, x)$ is definable, this means that it (and therefore $\text{SAT}(y, z)$, and therefore $\text{TRUE}(x)$) is not definable.

Definability and Computability

Showing that truth is not definable is already a very interesting result. Solving the above exercises should have convinced you that any property of natural numbers that can be decided by a purely syntactic manipulation of a mechanical sort is definable. Even if it didn't convince you, this is true under standard precisification of what "of a mechanical sort" means.

In fact, we can say something more specific. If a property or relation is definable by a formula where every quantifier is *bounded* (that is, they are all of the form $\exists x((x < t) \wedge \phi)$ or $\forall x((x < t) \rightarrow \phi)$), we say that the property or relation is Δ_0. If we can write the definition with a string of unbounded existential quantifiers followed by a Δ_0 formula, then we say that it is Σ_1, and for universal quantifiers we say that it is Π_1. If a property or relation can be defined both by a Σ_1 and by a Π_1 formula, then we say that it is Δ_1. (Similarly, prefixing universal quantifiers to a Σ_n formula gives a Π_{n+1} formula, and prefixing existential quantifiers to a Π_n formula gives a Σ_{n+1} one.)

Now, every Δ_0 relation is obviously Turing-decidable - since each quantifier has an explicit bound, a Turing machine can check whether or not the property holds by checking every value up to the bound.

Exercise 9 Show that all the properties and relations we have defined so far are in fact Δ_0. (This problem takes more time than it's worth to do the whole thing, though you might want to check a couple of cases.)

Now, we can repeat the same steps as before to show that, given a Turing machine M, the relation $MNEXT(x, y)$, saying that x and y are both the codes of a legal state of M, and that y is the next state of the machine after being in x, is Δ_0. Thus, using the sequence trick, we can show that $MHALT(x, y)$ is Σ_1, where $MHALT(x, y)$ says that machine M halts in state y on input x.

Thus, any recursively enumerable relation is Σ_1. Since a decidable relation is just one such that both it and its complement are recursively enumerable, this means that every decidable relation is Δ_1.

It turns out that a relation is Turing-decidable iff it is Δ_1, and recursively enumerable iff it is Σ_1. It is not too hard to show that there are Σ_1 relations that are not Δ_0, and similarly for every Σ_{n+1} and Δ_n. Thus, beyond saying that provability is definable, we can say much more specifically that it is Σ_1, while truth isn't definable by any formula, so the two notions are in fact quite far apart.

Provability

Exercise 10 Write down formulas showing that the following sets or relations are all definable (using only bounded quantifiers, if you feel like it):

- $\textsc{Axiom}(x)$, saying that x is the code number of a logical axiom (refer to the handout on the Completeness Theorem to see what counts as a logical axiom). It may be easier to tackle this in 15 or so separate steps - one for each type of logical axiom.
- $\textsc{ModusPonens}(x, y, z)$, saying that x, y, z are all code numbers of sentences and that z's sentence is a direct consequence of x's and y's using modus ponens.
- Assume that $\Gamma(x)$, saying that x is the code of a sentence in Γ, is definable (using only bounded quantifiers). Let T be the system of all derivations using only elements of Γ and logical axioms, with modus ponens as the only rule of inference. Show that $\textsc{GammaDerivation}(x)$, saying that x is the code of a sequence of sentences, which constitutes a derivation in T, is definable (and in fact Δ_0).

Gödel's important addition to Tarski's result was to do all the exercises in this handout and thus show that $\textsc{GammaDerivation}(x, y)$ is definable (and in fact Turing-decidable). Thus, $\exists x(\textsc{GammaDerivation}(x, y))$ states that y is provable in T.

Of course, to show that $\textsc{GammaDerviation}(x, y)$ is definable, we need to know what set Γ of assumptions we're working with — Gödel assumed that the set of axioms was decidable. This seems like a reasonable criterion — having a system of proof doesn't do much good if you can't even recognize *axioms* when you see them! Importantly, Gödel assumed very little else about the set of axioms. All he assumed was that it contained the following set of axioms, called the Peano Axioms, and was decidable:

1. $\forall x \neg(x < x)$
2. $\forall x \forall y \forall z(((x < y) \wedge (y < z)) \rightarrow (x < z))$
3. $\forall x \forall y((x < y) \vee ((x = y) \vee (y < x)))$
4. $\forall x(x < Sx)$
5. $\forall x \neg(0 = Sx)$
6. $\forall x \forall y((Sx = Sy) \rightarrow (x = y))$
7. $\forall x((x = 0) \vee (\exists y(x = Sy)))$
8. $\forall x((x + 0) = x)$
9. $\forall x \forall y((x + Sy) = S(x + y))$

10. $\forall x((x \cdot 0) = 0)$
11. $\forall x \forall y((x \cdot Sy) = ((x \cdot y) + x))$
12. $\forall x((xE0) = S0)$
13. $\forall x \forall y((xESy) = ((xEy) \cdot x))$

With a little work, you can see that for any sentence with no quantifiers and no variables, either it or its negation can be proved from these axioms. (We use axioms 9, 11, and 13 to reduce the right side of any operation to 0, then 8, 10, and 12 to remove those cases, and then we have an equation or inequality with two numerals, which can clearly be decided by axioms 1, 2, 4, 5, and 6. 3 and 7 are important only when using variables.) We certainly want our system to be at least that strong.

If we additionally assume that the axioms of system T are all true, then we know the set of provable sentences will be a subset of the true ones. Thus, since these two sets are distinct, we see that any true system T will be incomplete, in that there will be true sentences that it doesn't prove.

Consistency

By talking about a particular system T, Gödel also allowed consideration of consistency for this system. Since $\textsc{GammaDerivation}(x, y)$ says that x is a proof in T of y, and since we know that an inconsistent system has proofs of every sentence, we can write down a formula $\textsc{Con}(T)$ that says T is consistent, just by saying $\exists y(\textsc{Sentence}(y) \wedge \neg \exists x \textsc{GammaDerivation}(x, y))$.

Gödel's Second Incompleteness Theorem states that $\textsc{Con}(T)$ is always one of the unprovable sentences for any consistent system T. (He did this by showing that if T contains the above axioms, then it can prove the statement that all these above things are definable, and that truth is not, which would let it prove one of the sentences he showed was true but not provable.) In particular, if U is any system containing more axioms than T, then it is provable (using just the axioms in the set above) that $\textsc{Con}(U) \rightarrow \textsc{Con}(T)$. Thus, $\textsc{Con}(U)$ will also be unprovable.

Since just about every area of math is at least powerful enough to express and prove all these axioms, this means that no weaker system will ever be able to prove their consistency. This dashed Hilbert's hopes that some weak part of arithmetic could prove that all of set theory and analysis are consistent. Gödel showed that Hilbert was right to assume that some weak part of arithmetic could express these claims, but wrong to think that it could prove them.

One moral of the story is that axioms don't come for free — we can't prove them to be consistent. We just have to convince ourselves that they're consistent. It also means that mathematics can't be completely mechanized - there is no automated procedure that will decide whether or not a given sentence in the language of arithmetic is true (or even provable, as it turns out). There are procedures that can show certain sentences to be true, and that will eventually

list every provable sentence, but for things that aren't provable, we might never be able to show this fact.