Russell's Camelot and Gödel's Incompleteness

Larry Fenn

June 3, 2011

1 Introduction

Kurt Gödel published his article 'On formally undecidable propositions of *Principia Mathematica* and related systems' in the 1931 volume of *Monatshefte für Mathematik*.[1][2] The paper contained the exact opposite of what mathematicians who were working on the basis of math wanted to hear. Up until then, mathematicians had attempted to pin down the structure of mathematics so that it relies on as few assumptions as possible at its base. Hilbert's second problem from his landmark 23 problems in 1900 was 'prove that the axioms of arithmetic are consistent'. The seminal *Principia Mathematica* by Russell and Whitehead was a three volume work to establish a firm foundation for mathematics. The focus was to find a system of axioms and rules of applying them to establish a firm, purely logical basis, for all of mathematics that had heretofore been taken for granted at its lowest level. The dream was to prove the consistency of mathematics so that the nightmare scenario of eventually deriving a contradiction (making mathematics ultimately false) would be proved to be impossible. Gödel showed in his paper two important facts about axiomatic systems that sought to be the basis for a comprehensive foundation of mathematics: that there are statements that are true but unprovable, and perhaps even more surprising, that said system of axioms will lead to a proof of its own consistency (lack of contradictions) if and only if it is itself inconsistent.

2 Essential Utilities

The language of Gödel's proof is one of very formal logic. Thus it will be necessary to do some substantial clarification of his terms.

First, *PM* refers to *Principia Mathematica*, which is used in Gödel's paper as his example of a comprehensive system of axioms. A full discussion of what *PM* entails is beyond the scope of this paper, but for now it suffices to say that it is a system of axioms and rules of inference and a thorough documentation of how to construct mathematical truths (as opposed to falsehoods) from these assumptions. In addition to *PM* we also have the Peano axioms, *P*, which are a set of axioms that define the natural numbers.

While it may do to skim the three-volume, epic tome that is the *Principia Mathematica*, it will be necessary to describe P, Gödel's simplified version:

2.1 Basic Signs and Formulae

The *basic signs* of P:

Constant: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \forall, \exists, =, (,), 0$, S() (*the successor of*), $+$.

Variables:
- Type one: $x_1, y_1, z_1...$ (natural numbers, including zero)
- Type two: $x_2, y_2, z_2...$ (sets of the natural numbers)
- Type three: $x_3, y_3, z_3...$ (sets of sets of natural numbers)
- ...

The successor item bears some explanation. With the Peano axioms, 0 is given as a constant sign. Every other natural number n can be constructed by applying S() n many times — hence, S(0) is 1, S(S(0)) is 2, and so on. Note that these are all going to be variables of type one.

The basic signs allow for the construction of *formulae*: a formula is any construction of the basic signs with the usual syntax that results in a statement about a variable. An example is $\exists y$: S(0) + y = S(S(0)) (*there is a y such that 1 + y = 2*). Note that we are not yet concerned about the content (i.e. the truth value) of these formulae– simply in the syntax and ability to construct them. Note also that formulae are not allowed to be infinite in length– the notion that all formulae terminate will be very important later.

Any formula without arbitrary variables is a *proposition-formula*. For example, the formula 'S(0) + S(S(0)) = S(S(S(0)))' is a proposition-formula equivalent to 1 + 2 = 3. A formula with n free variables an *n-ary relation sign*. This complicated name just indicates that the statement indicates some statement about n different variables. In the case $n = 1$ we will use the special term *class sign*. Our example formula above about y is an example of a *class sign*.

Lastly, the class of *provable formulae* is the set that contains all the axioms and all the formulae that come from what Gödel calls 'immediate consequence'. Any given provable formula must have a proof. A proof can be thought of as a sequence of formulae, with the immediate consequence relationship linking them (implication would be another word for immediate consequence). Important to note is that the proof, additionally, must be finite in length: mathematics does not accept proofs of infinite length. Another term for this set of formulae for a specific set of formulae and axioms rather than the P case we are currently discussing is Conseq(κ), where κ is some set of formulae as well: Conseq(κ) is the set of all formulae that follow from a set of axioms or propositions.

2.2 Primitive Recursion

An important detail here is to be aware that we are also going to include in our axioms the principle of mathematical induction. It will be sufficient to establish that the principle of

induction here works very much the same as in all mathematics. If for some statement P the initial case is true, then if for the n case the $n+1$ case is true, then P is true for all n.

We can define a *primitive recursion* $f(n)$ by defining $f(\mathrm{S}(n))$ in terms of the values of the function $f(n)$. From this we can define a class of functions called *primitive recursive* functions. A *primitive recursive* function is given by one of three criterion:

1. The functions S (successor), Z (zero for all x), and I_i^k where $\mathrm{I}_i^k(x_1, x_2, ...x_k) = x_i$ are all considered primitive recursive.

2. Any composition f of primitive recursive functions g and h.

- Example: S(Z) is a composition of S and Z. Note that it is also a constant of value 1.

3. Any function f that can be defined from primitive recursive functions g and h by primitive recursion.

- Example: $f(\mathrm{S}(n)) = \mathrm{S}(n) \times f(n)$ where $f(0) = \mathrm{S}(\mathrm{Z})$ is the factorial function– here we have a successor function and a primitive recursion.

2.3 Consistency and Completeness

The notion of consistency and completeness are very important for this discussion.

Completeness is, for our purposes, the ability for a given axiomatic system to express all possible mathematical truths. In a sense, it reflects the idea that given a starting set of axiomatic propositions, every possible statement about arithmetic can be painted with a truth value: if it logically follows, then it would be true, else, false.

In addition, a given system of axioms is consistent if they do not express a contradiction. In a more formal sense, a consistent system is one where there doesn't exist any formula a where a and $\neg a$ are true results through separate proofs. This reflects the idea that no matter how deep our results from starting axioms get, we will never approach a statement that is somehow both true and false.

We can extend this sense to ω-consistency: A system is ω-consistent if there doesn't exist a formula a where $a(n)$ is true for all n and that there is some n where $\neg a(n)$ is true. Note that the n where $\neg a(n)$ is true may occur anywhere- it is only the existence of such an n that violates ω-consistency. The difference between ω-consistent and consistent is that ω-consistent $\Rightarrow$ consistent: a system that is ω-inconsistent does not have to indicate where the contradiction occurs in n, or whether or not it is ever even obtained- but merely that the contradiction does exist. The opposite is not necessarily true: in order to be ω-consistent we are concerned with the truth of the formula a for all n, not just at a single n. It may be that both $\neg a(n)$ and $a(n)$ can be proven true for some n, but if $a(n)$ is additionally not true for larger values of n then the statement does not violate ω-consistency. Lastly, inconsistent $\Rightarrow$ ω-inconsistent via the contrapositive.

With all that vocabulary out of the way, we are now in a position to understand the significant result of the incompleteness theorem.

2.4 The Theorems

The First Incompleteness Theorem: For every ω-consistent primitive recursive class κ of formulae there is a primitive recursive class-sign r such that neither $\forall v.r(v)$ nor $\neg\forall v.r(v)$ belongs to Conseq(κ)

The First Incompleteness Theorem (morally): A system of axioms cannot be both consistent and complete, since there will always either be a statement which is undecidable (inconsistent), or insufficiently strong (incomplete).

A very intuitive way to understand what is going on here is to consider a computer that has two specific rules:

1: Print out only true statements.

2: Print out all true statements.

Suppose such a machine exists, and outputs all that it can. Now consider what happens to the statement 'this statement will not be printed'. Either it will be printed or it won't be printed. If it will be printed, then the machine will have failed in its design and output a false statement (this is the analogue of ω-inconsistent). If it isn't printed, then the statement itself is true. However, without ever being printed then the machine will have failed its second design specification, to print out every true statement. Thus in either respect the machine will have failed. By applying this metaphor to arithmetic then we arrive at the conclusion Gödel reached, where any system of axioms that hopes to satisfy condition 1 by having only true statements must necessarily include statements that are true that cannot be proved from within the system itself.

The other result of this paper, Gödel's Second Incompleteness Theorem, is as follows.

The Second Incompleteness Theorem: If P is consistent, then the consistency of P is not provable from P.

In other words, no axiomatic system is strong enough to be able to prove its own lack of contradictions. I will not be going over the proof of this statement.

3 Proof of the First Incompleteness Theorem

The general method of the proof is as follows.

1: Show P can 'capture' all primitive recursive functions: we want P to be able to express all primitive recursive functions.

2: Establish as a primitive recursive function the function *Provable* such that *Provable* tells if a given formula is provable in P.

3: Craft a formula G such that G is true if it is unprovable in P.

3.1 P can express all primitive recursive functions

First claim: *Show P can 'capture' all primitive recursive functions: we want P to be able to express all primitive recursive functions.*

Proof: Begin with the list of qualities of primitive recursive functions have:

1. The functions S (successor), Z (zero for all x), and I_i^k where $\mathrm{I}_i^k(x_1, x_2, ...x_k) = x_i$ are all considered primitive recursive.
2. Any composition f of primitive recursive functions g and h.
3. Any function f that can be defined from primitive recursive functions g and h by primitive recursion.

Suppose that the following is true:

1. P can express S, Z, and the identity.
2. If P can express g and h, where g and h are primitive recursive, then P can express the composition f of g and h.
3. If P can express g and h, where g and h are primitive recursive, then p can express the function f defined by primitive recursion from g and h.

If so, then any primitive recursive function f which is built up from the three qualities of primitive recursive functions is built up using those three principles exclusively by chaining them together– therefore the ability for P to express the three basic qualities means that P can also express the result of chaining them together, hence f will be expressible in P.

To prove the first claim, it suffices to show the explicit formulae for the three initial functions.

S: $\mathrm{S}(x) = y$

- y is the successor of x.

Z: $\mathrm{Z}(x, y) = (defined)(x = x \wedge y = 0)$

- The second condition , y = 0, applies no matter what x is (x = x).

Identity: The identity function is a little more cumbersome, but for example, the identity function $\mathrm{I}_1^3(x, y, z) = x$ can be given by $\mathrm{I}_1^3(x, y, z, u)$ =(defined) (x = u $\wedge$ y = y $\wedge$ z = z)

- This formula is satisfied by a unique u for all the other variables (y = y, z = z): u = x.
- This formula expansion can work for any I_k^n.

To prove the second claim, represent g and h by G(x, y) and H(x, y). $f(x) = h(g(x))$ is going to be represented by $\exists$z(G(x, z) $\wedge$ H(z, y)). This is an explicit representation of the composition of g and h.

The last claim requires some ingenuity. Consider the factorial function:

- $0! = 1$
- $n! = n \times (n-1)!$

In a more general sense, we see that for any f defined via primitive recursion, we have:

- $f(x, 0) = g(x)$
- $f(x$,S$(y)) = h(x, y, f(x, y))$

If we let $z = f(x, y)$, then we have a sequence of numbers $k_0, k_1, k_2, ..., k_y$ where $k_0 =$ g(x), and if $u < y$ then $k_{u+1} = h(x, u, k_u)$, with $k_y = z$.

We now have to turn to consider a method of encoding a sequence inside P. Later on we will see a trick that involves prime factorization to encode sequences (and sequences of sequences!) into natural numbers. The problem with the prime factorization method is that P does not include the method for extracting the sequence, when given from a natural number. We will instead use another trick based in number theory: the Beta function. Let $\beta(c, d, i)$ =(defined) the remainder of c divided by $d(i + 1) + 1$. The utility of the Beta function is that, for a sequence of numbers of $n + 1$ length, there is a pair c and d such that $\beta(c, d, i) =$ the ith term in the sequence (the sequence starts with a zeroth term).

Claim: The Beta function can encode a sequence of $n + 1$ length starting at zero. $(k_0, k_1, k_2, ..., k_n)$

Proof: Let z be max$(k_0, k_1, k_2, ..., k_n)$. Let $d = s!$. Note for $0 \leq i \leq n$ the sequence defined by $d_i = d(i + 1) + 1$ are relatively prime–

Claim: The terms of the sequence $d_i = d(i + 1) + 1$ are all relatively prime.

Proof: Suppose otherwise: let j, k be the indicies of the terms which are not relatively prime, $j, k < n$. Let $k > j$ without loss of generality. Thus $dj + 1$ and $dk + 1$ share a prime factor p. This also means that $s!j + 1$ and $s!k + 1$ share a prime factor p, by our definition of d. Therefore $p > s$, since any number up to s will have remainder 1. p divides $dj + 1$ and $dk + 1$, so p must also divide $dk + 1 - (dj + 1)$, or $d(k - j)$. We already showed p doesn't divide d else it would not divide $dj + 1$, so p must divide $(k - j)$. However, $k - j$ must be less than n as per our definition of j and k above. Thus p must be greater than n, but also less than n, a contradiction. #

So all the d_i are relatively prime. There is a theorem in elementary number theory, the Chinese Remainder Theorem, that states that for any sequence D of relatively prime numbers, as c runs from 0 to |D|- 1 (|D|is the product of all the terms in the sequence), the sequences of the remainders of c divided by all the terms in D are all different from each other. The proof of the Chinese Remainder Theorem is in the Appendix in case the reader is unfamiliar with it. The Chinese Remainder Theorem tells us that as we go from $c = 0$ to |D|- 1 we will get every possible sequence of remainders. Thus we will have some c

such that the sequence given by the remainders of c divided by the terms in D will be the sequence $k_0, k_1, k_2, ..., k_n$. And so the Beta function (with corresponding c and d) exists for any sequence $k_0, k_1, k_2, ..., k_n$, $\beta(c, d, i) = k_i$. #

Now we return to where we had rewritten the definition of f as a sequence of numbers. We use the Beta function now, so that there is some c, d where $\beta(c, d, 0) = g(x)$ and if $u < y$ then $\beta(c, d,\mathrm{S}(u)) = h(x, u, \beta(c, d, u))$ and $\beta(c, d, y) = \mathrm{z}$.

Knowing g and h are already primitive recursive and that P can already express them (as assumed), then label them G and H inside P. Now we can transform the statement into one inside P:

$\exists c \exists d \exists k[B(c, d, 0, k) \wedge G(x, k)] \wedge (\forall u \leq y)[u \neq y \rightarrow \exists v \exists w(B(c, d, u, v) \wedge B(c, d, S(u), w)) \wedge H(x, u, v, w,)] \wedge B(c, d, y, z)$

This is explicitly the third quality of a primitive recursive function, inside P.

From this we can conclude that P can indeed express all primitive recursive functions. What remains is now to show that the function *Provable* (semantically, given a formula in P, *Provable* tells us if it is provable in P) is a primitive recursive function, and lastly to create a formula that carries the meaning 'this formula is unprovable in P'.

3.2 *Provable* is a primitive recursive function in P

The proof requires we establish the connection between proofs of mathematical statements and natural numbers. This is accomplished by mapping the basic signs of P to natural numbers in a specific way: Gödel outlines his method by corresponding the constant signs with the first 13 odd numbers and every variable with type n with a number of the form p^n, p is a prime that sequentially follows 43: The third variable of type 2 would therefore be the third prime after 43 to the second power. Through this method we have a one-to-one correspondence of the basic signs and the natural numbers.

Now we move on to corresponding sequences of the basic signs, or formulae, to natural numbers. Gödel utilizes the uniqueness of prime factorization to correspond a sequence of natural numbers $n_1, n_2, n_3, ...n_k$ to a number $2^{n_1} \cdot 3^{n_2} \cdot ... \cdot p_k^{n_k}$ with p_k is the kth prime. In this way, every sequence of natural numbers (formulae, now) has a one-to-one correspondence with a single (albeit very, very large) number. So the link between a formula and a single natural number has been established. To link proofs to single natural numbers, one needs only to repeat the procedure of taking prime powers. A proof can be thought of as a sequence of formulae, one following another with the relationship of 'immediate consequence' or 'implication' linking them. Thus, if we have a proof that consists of formulae $f_1, f_2, ..., f_n$ then the sequence of formulae (necessarily of finite length) can be associated to a single natural number again by $2^{f_1} \cdot 3^{f_2} \cdot ... \cdot p_n^{f_n}$.

With this in hand, we need to do several things before we can jump straight to messing things up with a 'this statement is false' trick. We are armed only with what we called the 'basic signs', knowledge of logical relations and certain properties of sets via the axioms, and the knowledge of primitive recursion. It needs to be established that mathematical concepts can also be encoded this way: everything from 'divisibility' to 'implies' to 'provable'.

Gödel's paper contained a laborious deduction through 46 steps on the primitive recursive nature of *Provable* by composition and recursion on the initial three functions. We will utilize

an argument provided by Peter Smith [3] as to the primitive recursive nature of $Provable$.

Begin with the concept of open-ended searches versus a bounded procedure. A bounded procedure is one that is guaranteed to terminate. A primitive recursive function is one that is specified by chaining together recursion and composition. Therefore we can argue that $Provable$ is primitive recursive. $Provable(x)$ depends on the function $ProofFor(x, y)$ which holds when x is the number corresponding to the proof of the formula coded in y. The process of decoding x and y is a bounded procedure– it hinges on factoring prime numbers and reversing the mapping given earlier. The process of checking if the proof is legitimate is also a bounded procedure– if each formula that consists of a step in the proof is either an axiom or a result of the immediate consequence operation, then the proof is legitimate (we are bounded by the length of the proof [and infinite proofs were ruled out]). Lastly, the procedure needs to check if the final line of the proof given by x's code is the formula in y's code: since our formulae, too, are bounded then we have another bounded procedure.

In every step of this process we have used a bounded procedure, one that terminates eventually. Therefore, we can conceive of writing this large (bounded) sequence of computations on a computer as a sequence of bounded 'for' loops only, without using a 'while' loop (and inviting the possibility of open-ended searches). Therefore the properties and relations from the sequence of 'for' loops we have encoded must depend only on the properties and relations of the basis of the computer we are using to compute all of the decoding, proof-checking and encoding: and this can be based on primitive recursive functions by our assumption that P is an axiomatic system that provides a basis for arithmetic itself.

Now if the $ProofFor(x, y)$ function is primitive recursive, then we will define $Provable(x)$ as: $Provable(x) \Leftrightarrow \exists y : proofFor(y, x)$ 'x is provable is equivalent to there is a proof for x'

And $Provable$ is going to be primitive recursive because $ProofFor$ is a primitive recursive function.

3.3 Crafting G

Let us define G such that the meaning of G is 'G is not provable in P'. The way we will go about this is to first add the last two functions to our tool-kit, Gdl and $diag$ (diagonalization).

Diagonalization is the process by which substituting in to a class-sign (one free variable) the number that itself encodes the class-sign. $diag(n)$ is defined as a function which, if n is the code for some formula with one free variable, gives us the code number for that formula's diagonalization. $diag$ is a primitive recursive function because of a simlar argument for $ProofFor$: all of the mechanics behind it are bounded search computations. The $Gdl(m, n)$ relation holds when m codes the proof of the diagonalization of n. Gdl is a primitive recursive function is a composition of $ProofFor$ and $diag$. Now, if we let U(y) be the function such that for all x $Gdl(x, y)$ is false, and we diagonalize U, we arrive at G. G is now a formula that itself was created inside the boundaries of P and primitive recursion, but itself says 'G is true $\Leftrightarrow$ G is unprovable in P': G = $diag$(U).

Proof of G's undecidability:

Substitute in the statement for U in G's definition: G = $diag$(U) tells us to put in place of y the number U$\prime$ that codes U into U itself: G = U(U$\prime$). U(y) = $\forall x \neg Gdl(x, y)$ from above,

so we now have G = $\forall x \neg Gdl(x, U\prime)$. G is therefore true only when there doesn't exist an x such that x is a code for the proof of the diagonalization of the formula given by the number U$\prime$ (direct translation of G's definition to English). However, U$\prime$ corresponds directly to U itself, and we let G be the diagonalization of U. Therefore G is true only when there doesn't exist a proof of G. Likewise, if G were provable, there would be a code for the proof and therefore G would be false. Therefore, G is true if and only if G is unprovable in P.

By our manipulations we have created a number, G, that 'says' via our encoding, 'I (G), am not provable in P'. Thus if we try to apply our arithmetic relationship $provable_\kappa(G)$ we will find that, if we assume that κ is both complete and ω-consistent, then we will have a contradiction; we will have violated our consistency: since if $provable_\kappa(G)$ is true, then G must be false- and if $provable_\kappa(G)$ is false, then G becomes true. This leads us to conclude that G must be a heretofore unheard of 'undecidable' statement.

4 Conclusion

An interesting footnote to this discovery of incompleteness is that the result will always apply to stronger axiomatic systems than P. Suppose we add a new axiom to P in the hopes of avoiding this undecidability boondoggle: if we have a set of axioms A and realize that G_A is some unprovable statement inside A, then we will create A + G_A and call it B. The problem here is that, in crafting G_A we only needed the key of 'unprovable in A'– so nothing stops us from crafting G_B with the semantic meaning of 'unprovable in B'. In general, if we come across an unprovable statement and we assume it is true, it will not make the system any more complete! Adding new axioms to κ does not change its essential incompleteness, because even if we artificially add in incomplete statements as we come across them to the system, one needs only to create the statement 'unprovable in κ + extensions'.

5 Appendix

5.1 Proof of the Chinese Remainder Theorem

Theorem: For any sequence D of relatively prime numbers, then as c runs from 0 to $|D|$- 1, the sequences given by the remainders of c divided by each term in D are all different from each other. *Proof:* Suppose not. Then there is c_1, c_2 such that $c_1, c_2 < |D|$, and the sequence of remainders given by c_1's division by the terms of D is equal to the sequence given from c_2. Let $c = c_2 - c_1$, so c ¡ $|D|$. If c_1 and c_2 both leave the same remainder after division by d, then c must evenly divide by d, and for all d_i in the sequence D. Since all the terms in D are relatively prime, this means that c must also divide by their product $|D|$. However, we let $c = c_2 - c_1$ so $c < |D|$: a contradiction. #

References

1. Solomon Feferman ed., On formally undecidable propositions of Principia Mathematica and related systems I, (1986). Kurt Gdel Collected works, Vol. I. Oxford University Press: 144-195

2. Kurt Gödel, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I, Monatshefte für Mathematik und Physik, **38** (1931), 173-98

3. Peter Smith, "An Introduction to Gödel's Theorems", Cambridge University Press (2007)