Kurt Gödel, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I.

Paper Title

*On formally undecidable propositions of Principia Mathematica and related systems I.*

Three points on this:

* *undecidable*: Note that this word has at least two different uses. Nowadays in CS we typically use it to mean Turing-decidable: i.e. we can encode a decision program on a Turing machine that will always halt with "yes" or "no". However, in Gödel's paper it is used in the earlier logical sense: a proposition A is decidable exactly when we can construct a proof of A or we can construct a proof of ¬A.
* *Principia Mathematica* refers to a book by Whitehead and Russell published in three volumes from 1910-1913. See [the SEP entry](http://plato.stanford.edu/entries/principia-mathematica/) or [Wikipedia](http://en.wikipedia.org/wiki/Principia_Mathematica). Not to be confused with Isaac Newton's *Philosophiae Naturalis Principia Mathematica* published in 1687.
* That "I" at the end: there is no part "II" to this paper: it was planned in order to elaborate some of the results, but felt to be not necessary (see footnote 68a on the last page). Sometimes people will refer to "Gödel's incompleteness theorems" (in the plural): both incompleteness theorems are contained in this paper as Theorems VI and XI.

Section One

|  |
| --- |
| **pg 145** |

The modern presentation of predicate logic (e.g. using natural deduction) assumes certain inference rules and conventions that were not available or as well-established in Gödel's day. Hence, in referring to the relatively-recent *Principia Mathematica* (PM) and the [Zermelo-Fraenkel](https://en.wikipedia.org/wiki/Zermelo%E2%80%93Fraenkel_set_theory) (ZF) axioms, Gödel is fixing what he means by the "related systems" of the paper title. The ZF axioms for set theory are still the standard set of axioms in this field.

|  |
| --- |
| **pg 147** |

Here Gödel reveals the core of his idea - that symbols, sentences and proofs can all be mapped to natural numbers. This should come as little surprise to a modern-day computer scientist, who is well-familiar with the idea of representing everything as binary numbers, as least in principle.

147[-2]: The concept of a "class" is coming through from PM: for the remainder of this section you can safely read "class" as "set".

We can read a "class sign" here as the *characteristic predicate* of a set of numbers. For example, "isPrime" might be the class sign for all prime numbers, so that when "isPrime" is applied to an actual number it would return 0 (false = not prime) or 1 (true = prime).

|  |
| --- |
| **pg 149** |

149[1]: Note that *R* is not strictly a class sign, but an enumeration of the class signs (say, in alphabetical order) - *R(n)* is the nth class sign. The notation used here is a bit unhelpful, so you might prefer to write this as *Rn*.

149[3]: here, [α;n] is an "is-a-member-of" formula that would be true if *n* is in the class named α and false otherwise.

149[8]: *Bew* = "beweisbare Formel" in German, "provable formula" in English.

The core assumption here is that we can actually express such a concept (and its negation) in the system: section 2 will show that this is indeed possible.

What follows then is a classic diagonalisation argument.

1. *K* is the set of numbers *n* for which *[R(n);n]* is not provable: that is, the set of all *n* where it is not provable that the number *n* is an element of the nth set.
2. Since *K* is then a set of numbers it must have some name in PM, say "S", which must be the qth set in our enumeration, for some number *q*
3. So, *[R(q);q]* asks if this *q* is in the qth set (i.e. the "unprovables", corresponding to *K*), and Gödel then asks if this formula is provable.

See Wikipedia on [Cantor's diagonal argument](http://en.wikipedia.org/wiki/Cantor%27s_diagonal_argument) for a reminder on what a diagonalisation argument in general looks like. Actually, the one used here (and later in this paper) more closely resembles [Cantor's Theorem](http://en.wikipedia.org/wiki/Cantor%27s_theorem), which states that the power set (set of all subsets) of the natural numbers is not countable.

149[-4]: The SEP discusses both the liar paradox and Richard's antinomy under [Paradoxes of Self-Reference](http://plato.stanford.edu/entries/self-reference/)

Note that there's a second (sensible) assumption here that anything provable is true - this is mentioned again on page 151. Section 2 will have to define *Bew* so that this is the case (definition 46, page 171).

|  |
| --- |
| **pg 151** |

The last paragraph of section 1 is worth thinking about - slowly!

Gödel asserts that, based on our definition, *[R(q);q]* is actually true (at the "metamathematical" level), since it is unprovable. This concept of "true" refers to our understanding of truth as we read a mathematical theorem (such as this one), without which it would be just a meaningless narrative. Note that the formal system under consideration (PM) talks about provability, not truth.

Section Two

|  |
| --- |
| **pg 151** |

Here Gödel starts to build up the language of PM. Most of this should be familiar.

Note the use of the letter ∏ to mean "for all", derived from an association between conjunction and multiplication (so Π is just conjunction/multiplication over an infinite set).

In part II of his definitions he inherits the strict type hierarchies from PM - this was intended to head off the "vicious circle" problems like Russell's paradox. Basically there's a hierarchy of types, with the numbers at level 1, sets of numbers at level 2, sets of sets of numbers at level 3 and so on.

Thus, for example

* type 1 includes individual numbers 0, 1, 2, 3, ...
* type 2 includes sets of numbers, like isPrime, isOdd, isGreaterThanFive, ...
* type 3 quantifies over sets of numbers, and so includes isEmpty, isSubsetOf, hasMoreElementsThan, ...
* type 4 quantifiers over sets of sets, for example isAPartition, haveNonEmptyIntersection, ...
* .....

|  |
| --- |
| **pg 153** |

153[2]: This just allows us to dispense with n-place predicates and deal with them as 1-place predicates whose arguments are tuples; a fairly standard construction from set theory.

153[14]: Thus we write *a(b)* to denote that the property a holds of b

153[18]: So Gödel writes xΠ(a) where we might write "∀ x. a"

153[-5]: The definition of substitution here should be non-problematic: note that type is preserved as *b* and *v* have the same type.

|  |
| --- |
| **pg 155** |

This page defines the basic axioms of the system. Note that Gödel allows himself some abbreviations

* a.b for "a and b", abbreviating ~(~a ∧ ~b)
* a ⊃ b for "a implies b", abbreviating (~a) ∨ b
* a ≡ b abbreviating a ⊃ b . b ⊃ a
* E x for "there exists an x", abbreviating ~(xΠ~ )
* = is just the usual "Leibniz equality". Footnote 21 refers us to ""PM (I,\*13)", i.e. Principia Mathematica, part I, chapter 13, where two variables are defined to be equal if no predicate (here x2) can distinguish between them.

There are then five groups of axioms:

**Group I:**

These are [Peano's axioms](http://en.wikipedia.org/wiki/Peano_axioms) (also called the Dedekind-Peano axioms) defining the natural numbers:

1. 0 is not the successor of any number
2. Two different numbers don't have the same successor
3. This is the axiom schema of induction. Note the (only) free variable here is x2 which is type 2, and thus a property of numbers.  
   In more modern notation we might write the property as "p", and express this as:
   * if p(0) and ∀ x.p(x) → p(x+1) then ∀ x.p(x)

**Group II:**

These are the axioms for propositional logic most commonly associated with [Hilbert's system](https://en.wikipedia.org/wiki/Hilbert_system) of the 1920s - natural deduction hadn't been invented yet. These should seem self-evidently true (though it's not obvious that this is the complete set). In reading them, implication binds weaker than disjunction.

**Group III:**

These are the basic axioms for predicate logic - note that the description immediately following the two axioms belongs to this as well. The first axiom allows you to instantiate a universal quantification (provided no variables get newly bound), the second allows you to pull the quantifier over *b* provided the quantified variable doesn't occur there.

**Group IV and V:**

These two axioms form the basics of set theory.

* Axiom 1 says that for any formula *a* there exists some predicate *u* such that satisfying *u* is equivalent to *a*. This is the axiom of comprehension which allows a set to be defined using a predicate. For example, if I can write "x1>3 ∧ x1<6" as a type-1 formula, then there should be some type 2 variable corresponding to this. In modern notation we would write this comprehension as {x1 | x1>3 ∧ x1<6}.
* Axiom 2 tells us how to define set equality: two sets (corresponding to *x2* and *y2*) are equal precisely when they have the same elements - that is, the characteristic predicates (*x2* and *y2*) both give the same answer for any putative element *x1*.

**Rules of deduction:**

155[-4]: Finally there are two rules of deduction which say:

* from *b* and *(~b ∨ c)* we can deduce *c*; if we remember that the latter is equivalent to *b ⊃ c*, this is just *modus ponens* (called "implies elimination" in natural deduction)
* from *a* we can deduce *(∀ v. a)*; this is just generalisation (called "forall introduction" in natural deduction).

|  |
| --- |
| **pg 157** |

This page defines a mechanism for translating formulas and sequences of formulas into natural numbers, known as the "arithmetization of syntax". This process has since become known as**Gödel numbering**, with the Gödel number of some formula *a* denoted here by *Φ(a)*.

The basic symbols are each assigned an odd number up to 13, and then the sets of variables (denumerably infinite for each type) are assigned to powers of primes; thus,

* the type 1 variables are given the numbers 17, 19, 23, 29, 31,...
* the type 2 variables are given the numbers 172, 192, 232, 292, 312,...
* the type 3 variables are given the numbers 173, 193, 233, 293, 313,...

Using this encoding, any formula is translated into a sequence of (odd) numbers in a unique way. It's not clear to me why we're only working with odd numbers, and later version of the theorem use the numbers 1,2,3,...

You might care to compare this process to that of tokenisation in a compiler, where the elementary symbols of the language are replaced with integer tokens. The slight difference here is the Gödel uses infinitely many tokens, since each variable gets a unique number.

There's one more trick here that's used to translate any **sequence** of numbers into a single unique number (lines 11-17). This uses the [Fundamental Theorem of Arithmetic](http://en.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic) (or Unique-Prime-Factorization Theorem), which tells us that any number greater than 1 can be decomposed uniquely into a product of powers of prime numbers.

Note that the Gödel number of even fairly simply formulas might be quite large, and decomposing into prime factors might not be an easy task: the important point here is that it is at least theoretically possible.

The importance of formulas-as-numbers is hinted at (from line 157[-12] onward): statements about formulas, like "is an axiom" or "is a provable formula" now become statements about numbers, and are thus (in theory) expressible within the language. Of course, now we have to show just how such statements can be expressed.

157[-4]: Here starts a "parenthetic consideration" that leads to the main set of definitions behind the theorem. Here, Gödel defines what it means for a function to be *recursive*: basically this means that it is defined by induction over its first argument, along with a few other standard operations.

* Note that what Gödel calls "recursive" here is nowadays called **primitive recursive**, and does not include the full set of computable functions.

|  |
| --- |
| **pg 159** |

159[1:4] Here the function Φ is being defined by induction over the first argument. The **base case** is denoted by the function ψ, and the **inductive/recursive case** is denoted by the function μ. Note that the inductive case has access (through its arguments) to three kinds of information: the value of *k*, the result of the computation for *φ(k,...)*, and, of course, the remaining arguments.

A relation is recursive if it can be implemented using a recursive function (here an answer of 0 denotes that the relation *does* hold).

**A note on notation:**

* In the centre of page 159 Gödel shows how to define a relation in terms of a primitive-recursive function, and via footnote 29, links the syntax to a book by Hilbert and Ackermann. In this syntax the, "~" symbol means "if and only if".
* Note that this symbol, along with "&" for conjunction and the overbar for negation

are symbols in the *meta-language* of the theorem itself. The previously defined symbols, including "~" for negation (pg 153) are symbols in the *object* language, PM.

* The square brackets used around φ here are just a variation in notation, and have no special meaning other than for grouping. For yet more complicated expressions (e.g. pg 168, point 8) curly braces are used, again for grouping.

**Theorems I-III** are broadly unremarkable, and just show how the basic elements of logic correspond to arithmetic operations. Theorem II sets up the logical connectives and III gives equality.

**Theorem IV** shows how three kinds of quantifiers can be defined as primitive recursive functions. The three quantifiers are:

* (Ex)[.....] there exists an *x*, such that .....
* (x)[.....] for all *x*, we have .....
* εx[.....] the smallest value of *x* for which .....

The first two yield logical statements (true or false) whereas the last yields a number.

All of Gödel's definitions here are (deliberately) "constructive", in that we can readily see how these functions might be implemented. Most importantly, note that each of the three quantifiers defined here is a **bounded quantifier**: there is always has an *upper bound* on the quantification (represented by φ). Thus we can immediately see how to implement them by, say, a simple for-loop.

|  |
| --- |
| **pg 161** |

161[-8] The function χ defined here is basically a loop that works out the smallest value for which *R* holds. If you look forward to the actual use of χ (at 163[5]) you can see that its used to define the ψ function, and that its argument is the upper bound for the search (as in 161[4]). Thus, a sensible way to implement this is to count down from the upper bound until we find a value that *R* holds for: this is the job of χ.

The base case for χ makes sure that we always return at least 0. In the recursive case, the role played by *a* is basically the guard in an if-then-else (since *a* and thus *α(a)* will be 0 or 1).

A Java-esque version of the χ function might look like:

import java.util.List;

public abstract class ChiFunction

{

// Really, this should be called rho...

abstract boolean R(int firstArg, List<Integer> otherArgs);

int chi(int upperBound, List<Integer> args)

{

if (upperBound == 0)

return 0;

// else the upperBound is n+1 for some n

int n = upperBound-1;

int prev = chi(n,args);

if ((! R(0,args)) && R(n+1,args) && prev==0)

return n+1;

else

return prev;

}

}

|  |
| --- |
| **pg 163** |

163[5:6]: One ψ is defined as a direct call to χ with the upper bound, defining *S* is as easy as checking that the value returned by ψ is actually a found-value and not the default value of 0 (i.e., that the relation *R* actually holds for the returned value).

There now follows a long sequence of definitions of primitive recursive functions, aimed ultimately at being able to define the concept of "provable" in the system. Given that the encoding of previous pages was based on powers-of-primes, it shouldn't come as a surprise that Gödel starts off by defining concepts such as divisible, prime etc.

The abbreviated names of the functions that Gödel defines are based on German words - in most cases this fortunately is the same as the English. In case it helps, I've listed some of the non-obvious terms, along with what I guess to be their translation:

|  |  |  |  |
| --- | --- | --- | --- |
| # | Abbrv | German | English |
| 6. | Gl | "Glied" | term |
| 10. | E | "Einklammerns" | enclosing within parentheses |
| 17. | Z | "Zahlzeichen" | numeral |
| 22. | FR | "Reihe von Formeln" | sequence of formulas |
| 24. | Geb | "Gebunden" | bound |
| 28. | St | "Stelle" | place |
| 29. | A | "Anzahl" | number |
| 34. | Z | "Zahlen" | figures (numbers) |
| 43. | Fl | "Folge" | consequence |
| 44. | Bw | "Beweisfigur" | proof array |
| 45. | B | "Beweis" | proof |
| 45. | Bew | "Beweisbare" | provable |

Many of these are only used internally here: the most important ones are:

* 15. *Gen*: the "for all" quantifier
* 17. *Z*: the numeral (f(f(f(... (0)...))) corresponding to a number
* 31. *Sb*: substitution of a value in for a variable
* 43. *Fl*: immediate consequence
* 45. *B*: is a proof of

|  |
| --- |
| **pg 171** |

171[9]: Definition 46: Note that the relation *Bew* ("provable") is **not** (primitive) recursive since there is no bound on the existential quantifier *(E y)*. There's no bound since we cannot think of a way to calculate an upper bound on the Gödel number of the proof, *y*, based on the formula to be proved, *x*. (If we were able to calculate such a bound from *x*, we'd have solved the *Entscheidungsproblem*).

Gödel has just defined 45 (primitive) recursive relations using the notation he has devised for this paper. In Theorem V he now asserts that any such (primitive) recursive relation can also be encoded in the language of PM. Here, *R* is a (primitive) recursive relation, e.g. any of 1-45 above, and *r* is its encoding in the language of the PM. The theorem asserts that, given any *R*, such a corresponding relation *r* can always be built. Equations (3) and (4) assert that *r* works in a way that corresponds to *R*: i.e. when *R* holds for some values *x1,...,xn*, then we can prove that *r* holds for these values or, more correctly, for the Gödel number of these values *Z(x1),...,Z(xn)*.

The proof is by structural induction on the degree of the function ("degree" is defined at 159[11]): basically on the number of other functions used in its definition.

|  |
| --- |
| **pg 173** |

173[10]: The definition of ω-consistent perhaps requires some disentangling: here ω is standing for "the natural numbers". Since *Flg(κ)* contains the consequences of *κ*, membership of this set means, essentially, "provable from *κ* ". The definition of ω-consistency says that it is **not** the case, for any *a*, that (with some abuse of notation):

* on the one hand *a(n)* is provable for all specific numbers *n*
* on the other hand *~(v Π a)* is provable.

As Gödel notes, this is not as strong as "proper" consistency (which states that you cannot prove both *A* and *~A* for any formula *A*). The theorem [was extended](https://en.wikipedia.org/wiki/Rosser%27s_trick) to cover full consistency by J. Barkley Rosser in 1936.

This definition now sets us up for Theorem VI, the "first incompleteness theorem" which starts at 173[16] and finishes at 177[16]. To prove incompleteness, Gödel needs to construct a formula *A* such that neither *A* nor *~A* are provable. As he notes in the statement of Theorem VI, The formula he constructs will actually have the form *v Gen r* (this is OK: he only needs to find one formula).

173[16]: The proof starts by extending the primitive recursive definitions of provability in PM (44, 45 and 46) to provability in PM plus any set of axioms *κ*. Equation 7 just says that if *x* is provable from κ then *x* must be an element of the consequences of κ. Equation 8 just says that assuming some set of axioms κ doesn't invalidate any proofs that were true without them.

|  |
| --- |
| **pg 175** |

This is probably the most important page of the proof!

On this page you can think of *x* as being the Gödel number of some proof array (i.e. the 'listing' of a proof), and *y* as being some formula that we want to prove.

We recall here from page 157 that 171 and 191 are the Gödel numbers for the first two variables of type 1. You can think of *17* and *19* as corresponding to *x* and *y* in this proof.

*Q(x,y)* thus means that: *x* is **not** a proof of the formula we get from *y* if we replace its second free variable with (the numeral) for *y*.

Thus, this equation 8.1 is the basic template for the diagonalisation mapping: we are making *y* not quite refer to itself, but rather to the Gödel number for itself.

The next step is to encode *Q* into the object language of PM. We know we can do this since *Q* is primitive recursive (as Gödel reminds us, this is Theorem V). Hence we can define some corresponding relation sign *q*, and Theorem V gets us from 8.1 to equations 9 and 10. As expected, this *q* represents a two-place relation with two free variables.

Equations 11 and 12 then give us two specialised version of the relation *q*:

* Equation 11 universally quantifies over the first variable (i.e. the variable *x* in *Q(x,y)*), saying thus that there is no *x* that is the proof of *y*.   
  Roughly, *p(y)* states that " *y* is not provable"
* Equation 12 gives us a hard-wired version of *q* that talks about *p*.  
  Roughly, *r(x)* states that " *x* is not a proof of *p* "

With the potential for self-reference (via Gödel numbers) now in place, we just need to spring the trap by applying *p* to the (Gödel number for) *p*.

The resulting term "17 Gen r" is the one Gödel was looking for: this is the undecidable sentence. It says roughly "for all *x*, *x* is not a proof of *p* ", or in other words " *p* is not provable". However, since *17 Gen r* is actually a version of *p*, it is as close as we'll get to a statement saying "I am not provable" (or "forall *x*, *x* is not a proof of me").

It might help to think of this in terms of sets (or "classes"):

* Equation 11 defines *p* as the set of (Gödel numbers of) all unprovable statements
* Equation 12 defines *r* as the set of numbers that are not proofs of *p*

Equation 13 then asks whether the Gödel number of *p* is in the set defined by *p*. Then working through the definitions we end up with *17 Gen r*, basically saying that no Gödel number is a proof of *p*.

Gödel then brings this back to equations 9 and 10, asking what would happen if we plug in *Z(p)* in place of *Z(y)* here. Equations 13 and 14 help us reduce down the resulting formula, to get equation 15 from 9 and and 16 from 10. Note that in both equations 9 and 10 we have *y* on the left of the implies and *q* on the right. However, in 15 and 16 we have worked this around, so now we have *r* on both sides, giving us the contradiction.

|  |
| --- |
| **pg 177** |

Gödel now completes the proof by reasoning about the κ-provability of *17 Gen r* using a straight proof-by-contradiction structure, familiar from other diagonalisation arguments.

The comment that the proof "is constructive" follows from the fact that Gödel has not just proved the theoretical existence of some undecidable statement *A*, but has actually shown us how to build such a sentence, namely *17 Gen r*. In theory, you could follow the rules of this paper and actually calculate the Gödel number of the undecidable sentence *17 Gen r*.

|  |
| --- |
| **pg 179** |

Here Gödel considers the implications if he weakens his premises to full consistency rather than ω-consistency: the consequence is that he can only prove ω-incompleteness rather than full incompleteness. Part I at the top of page 177 still holds, so *17 Gen r* is not provable, thus its negation can be added to the axioms without contradiction. However, since each instantiation of *r*is provable, this means that he has constructed a ω-inconsistent extension.

Section Three

Here Godel moves on to consider some other systems where his result also holds: we can replace recursive function with arithmetical functions (Theorem VII) and we can replace the full calculus (of page 155) with the restricted function calculus (Theorem IX).

|  |
| --- |
| **pg 181** |

Starting in section 3: These restrictions limit his language to talking about numbers with just addition and multiplication as the built-in functions. This "arithmetical" language seems much weaker than P since it is missing recursive function definitions entirely (so addition and multiplication have to be provided), but it *does* have unbounded quantification.

|  |
| --- |
| **pg 183** |

Theorems VII (stated here) and VIII on the bottom of page 185 form a pair, with VII setting up VIII: there are undecidable propositions even when we are limited to constructing functions using only 'arithmetical' operators. Since we have addition, multiplication and equality, this means that, for example, polynomial equations with natural number coefficients are formulas in this language.

He needs to show that he can actually re-create any recursive relation definition in this language of arithmetic, and he proves this by structural induction over the size of the recursive definition that you're trying to encode. He identifies two cases: the recursive relation is defined by

* Case 1: substitution via *ρ* and *χ1...χn*, or
* Case 2: induction via *ψ* and *μ* (case 2).

The first case is easy since the inductive hypothesis converts *ρ* and each *χi* into *R* and *Si*.

To deal with the second case he can get a good deal of the way using the induction hypothesis: basically to statement (17), which says there must exist some sequence of numbers *fi*, each one corresponding to the result of a previous 'iteration' of the recursion. However, the formula *(E f)...* is a problem, since this is a quantification over function-sequences, and he must convert this to something arithmetical.

|  |
| --- |
| **pg 185** |

The goal here is to convert formula (17) from the previous page into the arithmetical language, which boils down to converting a sequence of numbers (*f* here) into an arithmetic function. Gödel's trick here is to use the [Chinese remainder theorem](https://en.wikipedia.org/wiki/Chinese_remainder_theorem) to encode a sequence of arbitrary length into an arithmetic function of two numbers.

To encode a sequence of length *k*, Gödel needs to form a set of *k* equations of the form

* *n = fi mod ai*

where *i* runs from 0 up to *k-1*. The Chinese remainder theorem says that we get a unique solution for *n* if each of the *ai* are relatively prime. Gödel then generates the sequence *ai* using the formula *1 + (i+1)l!*. The '1+' at the start of this formula ensures that it can't be divisible by *l!*, and Godel uses this to show that no two *ai* can have a common divisor, as required (proving Lemma 1). This encoding has become known as [Godel's β-function](https://en.wikipedia.org/wiki/G%C3%B6del%27s_%CE%B2_function), since this was the name he gave it in a subsequent (1934) paper.

Gödel has now expressed (a given finite prefix of) the sequence of numbers *fi* as an arithmetical function of two numbers *n* and *d* (the latter is just *l!*), which allows him to substitute them into equation (17). Now, however, instead of '(E f)' he has '(E n,d)' where *n* and *d* are numbers, and the formula is arithmetical, as required.

Theorem VIII follows immediately from this.

|  |
| --- |
| **pg 187** |

187[1:10] Theorems IX and X here show that the incompleteness result applies to the restricted function calculus (rfc) as well as the general 'PM' calculus. Note that quantification in the rfc is limited to individuals (not classes), but not necessarily to numbers. It is basically what we now call 'first-order predicate logic', and was described in a textbook by Hilbert and Ackermann that Gödel cites.

187[15] Technically there should be no function symbols in the rfc (as it was originally defined). However, Gödel has a trick to allow function-symbols to appear in terms, and he calls a formula that use this syntactic extension "a formula in the extended sense" (ies).

187[-5] To deal with function symbols he uses a technique that goes right back to PM (Part I, chapter 14). Fist of all he notes that any function taking *n* arguments can be made into a relation with *n+1* arguments, where the extra argument is just the result. Thus *z = φ(x)* is the same as constructing a corresponding relation *F* and saying that the relation *F(z,x)* holds. However, to use this term in larger ones he needs some way of extracting the *z* out of this (since the result of *F(z,x)* is just true or false)- that's what the ι (iota) operator is for.

According to PM, something like *(ιz)F(z,x)* should be read as "the term *z* which satisfies *F(z,x)* ".

187[-2] "the method used in PM(I,\*14)" : to then get rid of the ι operator, we introduce an existentially-quantified variable to represent the function's result. As an example,

* if we started with the formula *x+y > 5*
* we'd replace the function + with a predicate, so *sum(z,x,y)* would mean "x+y = z"
* we'd then use the ι operator to put this back into the original formula as *((ιz)sum(z,x,y)) > 5*
* last, we'd use an existential variable to get rid of the ι, to get *(E b) sum(b,x,y) ∧ b > 5*

We end up with a formula in the ordinary predicate calculus (without function symbols). Note that there are no constant symbols in the rfc either, so we'll have to fake a zero if/when we need one.

|  |
| --- |
| **pg 189** |

So, Gödel wants to show roughly that: given any recursive function *F*, he can build a rfc formula that is satisfiable when *F* is true.

If *F* is recursive then we must have a recursive definition for it - i.e. it must correspond to some function Φn which is defined in terms of a bunch of other functions Φ1 ... Φn-1. Thus points 1, 2 and 3 here are essentially a statement that the recursive function definition Φn must be defined in terms of induction, substitution, or must be a numeral. The first two (equations 18 and 19) are similar to the argument in Theorem VII page 183, with the addition of equation 20 to deal with numbers as a special case (Theorem VII only worked with numbers anyway).

Equation 21 makes sure that Φ1 actually works like the successor function (see I.1 and I.2 on page 155). Equation 22 just asserts that Φn (the function being defined) is always zero, since the predicate *F* that it corresponds to is always true.

Note that, for any given recursive relation *F*, Φ1 ... Φn are actual, known function names in P: they are simply the function names that you have used when writing out the recursive definition for *F*.

Gödel now must build a formula in the rfc that is satisfiable when equations 18-22 are true. He takes these equations and substitutes each function name Φi (from system P) by a rfc function variable φi, and the number 0 by some new variable he calls *x0*. He now has a formula expressed in the rfc, and he just needs to show that it is satisfiable when the original (equations 18-22) is true.

|  |
| --- |
| **pg 191** |

This is an if-and-only-if statement, so we must prove it both ways.

The first case is trivial: we can, of course, find an appropriate instantiation for the function variables φi, since we can just pick the actual, known functions Φi in each case.

The second case is a bit more interesting. If we know the formula is satisfiable, then we know that there exists something corresponding to the variable *x0* and the function variables φi. He gives these "solutions" the names *0* and Ψi: think of these as functions "in the real world" that correspond to the function-symbols used in the rfc.

Gödel now builds a model that is exactly isomorphic (one-to-one) with the original one. He takes the set consisting of *a, Ψ1(a), Ψ1(Ψ1(a)), ...*, and then plugs the other Ψi functions into equations 18-22. Since these equations work for this new model, and this model's elements have a one-to-one map with the original functions (and the natural numbers), they must hold for these too.

Section Four

|  |
| --- |
| **pg 191** |

Section 4: This part of the paper looks rather like a postscript, but it contains the other fundamental result: that a system *P* (of the usual kind) cannot prove its own consistency. Here Gödel plays on two versions of what is means for a system to be consistent:

* there does not exist any formula *A* where you can prove both *A* and *¬A*.
* there exists some formula that you cannot prove (see footnote 63).

The second property follows from the first: if you could prove both *A* and *¬A* then you have a contradiction, and you can logically infer any result from this (thus *everything* would be provable).

|  |
| --- |
| **pg 193** |

Gödel will prove this theorem by showing that the first incompleteness theorem (Theorem VI) can be encoded in his system. The final part of his proof of Theorem VI has two propositions, and the first of these on page 177 is that (if the system is consistent) " *17 Gen r* is not κ-provable". He restates this in his notation in equation 23. Getting to equation 24 is basically an unwinding of equation 23: rather like going backwards through the reasoning on page 175.

It is interesting to note that it is only here that Gödel observes that his proof infrastructure, including the recursive functions and Theorem VI itself, can be encoded into system *P*. In particular, equation 24 is then encodable in system *P* and ends up as the statement *w Imp (17 Gen r)*, for suitable *w*.

Since Theorem VI is provable in system *P*, it results, including *w Imp (17 Gen r)*, must be provable in system *P*. Here's the catch: if *w* was also provable in system *P* then we could just apply modus ponens (defined at the bottom of page 155) to prove *(17 Gen r)* in system P, which is just what theorem VI says we can't do.

Thus we cannot prove *w*, a statement that the system is consist, in the system itself.

|  |
| --- |
| **pg 195** |

Gödel makes a few remarks here on the more general applications of his result.

In the 1920s there was a notable dispute between the Hilbert school and that of [L. E. J. Brouwer](https://en.wikipedia.org/wiki/L._E._J._Brouwer) over the use of constructive principles - hence Gödel's insistence on highlighting the "constructive" nature of his proof. The dispute is sometimes called the [Brouwer–Hilbert controversy](https://en.wikipedia.org/wiki/Brouwer%E2%80%93Hilbert_controversy), although Einstein called it a "frog and mouse battle".

Gödel also includes some consolation for Hilbert's programme, since consistency proofs were a major goal here. Gödel notes that he hasn't ruled out the possibility of a consistency proof, just the possibility of a system proving *its own* consistency. The degree to which this demolishes Hilbert's programme is still a matter of debate: see the [SEP entry](http://plato.stanford.edu/entries/hilbert-program/#4) for a quick summary.

Finally, Gödel adds a postscript (to this translation of his paper, which he had reviewed) noting his acceptance of the work of Turing. There's a nice (reasonably non-technical) review of Gödel's attitude to the work of Turing and Church in a paper by Martin Davis called [Why Gödel didn't have Church's thesis](http://dx.doi.org/10.1016/S0019-9958(82)91226-8) (*Information and Control*, 54 (1–2), 1982).

|  |
| --- |
| Logicomix |

This is the moment as re-imagined by pg. 286 of *Logicomix: an epic search for truth* by Apostolos Doxiadis and Christos H. Papadimitriou. That's meant to be von Neumann saying "it's all over" in the last frame.