Versions of Gödel's incompleteness Theorem
Ron Maimon, obtained from

`http://mathoverflow.net/questions/72062`

Includes parts of the Wikipedia article "Undecidable problems"

Edited and augmented by Armando Matos

January 15, 2014

Contents

[...] The construction in Gödel's theorem is often obscured by the heavy coding involved. Since today, coding is standardized in computer science, I prefer to state the construction explicitly as a computer program, instead of as a coded statement of first-order logic. Two proofs are the same when they construct the same computer program.

There are exactly three types of unpacked proofs of Gödel's theorem and related results, as far as I know. To save typing, a program P "runs" iff it does not halt.

Chapter 1

TYPE I

Self referential Π_1^0 statements (statements about the non-halting of a certain computer program)

1.1 GÖDEL-1

To prove Gödel's theorem Gödel's way (as clarified by Turing and Kleene), given an axiomatic system S whose deduction system is computable, you construct the program GÖDEL which does the following:

GÖDEL:

1. Print its own code into a variable R. (This is possible since you can write quines, and make quining into a subroutine)
2. Deduces all consequences of S, looking for a proof in S of the statement "R does not halt" ("R runs"); (this is a statement of arithmetic of the form "$\forall n F^n(R)$ is non-halting", where F is a primitive recursive instruction set for any computer you care to code up).
3. If such statement is found, halt.

The statement G – "GÖDEL runs" – is true precisely when S does not prove it. So G states "S does not prove G". The self reference is obvious in the first step of the program. This is equivalent to Gödel's original construction.

From the construction, you can read off the requirements on the axiom system. In order to be sure that GÖDEL halting leads a contradiction, the axiom system has to be able to prove every statement of the form "program P leads to memory state M after time t" for all integer times t, and for all programs P. Each of these statements is a finite computation, a Σ_1^0 statement, so the axiom system must be able to prove all true Σ_1^0 statements (or even just a subset of these rich

enough to allow a computer to be embedded in the language).

1.2 GÖDEL-2

Note that if S is inconsistent, it proves every statement, including "GÖDEL runs", so "S is inconsistent" implies "GÖDEL halts". "GÖDEL halts" also implies "S is consistent", if S can prove that it is Σ_1^0 complete. So the unprovability of "GÖDEL halts" is tantamount to the unprovability of $\mathsf{consis}(S)$.

But S can (falsely) prove "GÖDEL halts", without any contradiction, so long as GÖDEL never actually halts. This is saying that it is possible for an axiom system to prove its own inconsistency without actually being inconsistent, just by telling lies about computer programs. The assumption that S is omega consistent (or even just Σ_1^0 sound), means that it does not prove "P halts" unless P actually halts. So the same construction of GÖDEL proves the second incompleteness theorem as stated by GÖDEL, an omega-consistent system (or a Σ_1^0 sound system) cannot prove its own consistency.

The proofs of Gödel's theorem which go through the halting problem all give this construction.

1.3 ROSSER

The program ROSSER is just a slight modification of GÖDEL. ROSSER does this:

ROSSER:

Print its code into a variable R

Look in S for a proof of
(1) "R prints to the screen" or of
(2) "R does not print to the screen".

If (1) is found, halt without printing, if (2) is found, print "hello" and halt.

Now note that a consistent S cannot prove 1. nor 2., because either way, there is a halting computation that contradicts the statement. So a consistent S is incomplete. If we call the statement "ROSSER prints to the screen" by the name R, and its negation 2. by the name "notR", then "ROSSER does not print" is iff equivalent to "S proves R before notR", which is the standard gloss for ROSSER's construction.

ROSSER's statement is different than GÖDEL statement, because the statement "ROSSER does not print" is not equivalent to the statement "S is con-

sistent". But since the slightly different statement "ROSSER does not halt" actually is equivalent to "S is consistent", ROSSER's construction includes GÖDEL's construction in a simple way.

Here are some simple modifications which also prove Gödel's theorem:

1.4 PROOF-LENGTH

Given a provable statement of length L bytes in an axiomatic system S, there is no computable function of L, $f(L)$, which bounds the length of the proof of L (relatively short theorems can have enormously long proofs).

Construct PROOF-LENGTH to do the following:

PROOF-LENGTH:
- Print its own code into a variable R.
- Look through all deductions of S of length up to $f(|R|)$ bytes for a proof of "R prints `ok`".
- If such a deduction is found, halt.
- If not, print "`ok`" and halt.

In this case, the construction is clarified with a gloss: suppose $f(L)$ exists, then you can decide the halting problem by running through all proofs of length $f(|\text{"P halts"}|)$ for a proof of "P halts". If you don't find it, then P doesn't halt.

This is also a proof of Gödel's theorem, since if S is complete, then it will decide all statements of the form "P halts", and then you can compute the function f which is the length of the proof of the statement. But the program constructed is essentially the same as GÖDEL (actually ROSSER, in the version I gave here).

But the explicit construction does give you an important corollary: if you assume S is consistent, then just by the form of PROOF-LENGTH, you can see that PROOF-LENGTH has to print "`ok`" independent of the function f, since if it does not, this means it has found a proof that it didn't. So the assumption of $\mathsf{consis}(S)$ will collapse this f dependent enormously long proof to a short f-independent proof of the same statement.

This construction is just a finitary version of the original GÖDEL program, and this theorem is called the GÖDEL speedup theorem. The assumption of $\mathsf{consis}(S)$ reduces the length of proofs of certain statements by an amount greater than any computable function of the length.

1.5 LOB

Given an axiomatic system S, consider the program LOB which, given statement A, does the following:

LOB:
Input A
- Print its code into R.
- Deduce consequences of S, looking for "R halts implies A".
- If found, halt.

Let the symbols "$\vdash$", "$\downarrow$", "$\downarrow$", "$\Rightarrow$", and "$\iff$" denote "proves", "halts", "diverges" (does not halt), "implies", and "if and only if", respectively.

"LOB$\downarrow$" only if S proves "LOB$\downarrow \Rightarrow A$", and then S also proves "LOB$\downarrow$", so it proves A by modus ponens, so LOB$\downarrow$ iff A. But "LOB$\downarrow$" is equivalent to

$$(S \vdash \text{LOB}\downarrow) \Rightarrow A$$

and therefore to

$$(S \vdash (S \vdash A) \Rightarrow A).$$

Therefore, S proves

$$S \vdash ((S \vdash A) \Rightarrow A) \Rightarrow (S \vdash A).$$

This theorem can be repackaged into an infinite sequence of ever more obscure statements, by replacing "LOB halts" with its different equivalent forms (some of which contain itself), and eventually closing the recursion. The full set of Lob statements is generated by a simple recursive grammar.

LOB's theorem does not prove Gödel's theorem, but it extends it. The proof is of a similar kind.

1.6 TWEEDLEDEE and TWEEDEDUM

Consider the programs TWEEDLEDEE ("DEE" for short) and TWEEDLEDUM ("DUM" for short):

DEE:
- Print DEE's code into ME, and DUM's code into HE.
- Look for (proofs of) "ME$\downarrow$" and "HE$\downarrow$".
- If found "ME$\downarrow$", halt.
- If found "HE$\downarrow$", print "tweedle-dee-dee!" and go into an infinite loop.

DUM:
- Prints DUM's code into ME, and DEE's code into HE.
- Looks for (proofs of) "ME↓" and "HE↓".
- If it finds "ME↓" it halts, if it finds "HE↓" it prints "tweedle-dee-dum!" and goes into an infinite loop.

These give a kind of splitting theorem for axiomatic systems which satisfy the hypotheses of GÖDEL's theorem. "DEE↑" and "DUM↑" are both unprovable in S, since proving either one leads to a contradiction. "consis(S)" implies "DEE↓ & DUM↓", and conversely "DEE↓ & DUM↓" implies consis(S).

So if S is inconsistent, then one of DEE or DUM has to halt. But which one? This is not decidable in S. That is, S+"DEE↓" is a theory which is strictly stronger than S, since it proves "DEE↓", but is weaker than S+consis(S)" because it cannot prove "DUM↓".

To prove this, note that S proves "DEE↓ or DUM↓", i.e. "(DEE↓) ⇒ (DUM ↓)". So if S also proved "DEE↓ ⇒ DUM↓", it would prove plain old "DUM↓", which is impossible.

The reason for the spurious print statement is just to make absolutely sure that the programs DEE and DUM, which are so similar, don't end up identical, which would wreck the proof (this subtlety is hard to see if you don't unpack the construction into an explicit program, but it is also easy to avoid by using different variable names, or extra spaces, or whatever).

This construction is strictly stronger than GÖDEL's. It shows that for any sound system S, the implication "DEE↑ implies DUM↑" is unprovable. The construction provides a proof of Gödel's theorem, although it is similar to ROSSER (The statement "DEE↓" is provably NOT the negation of "DUM↓", that's the whole point)

I wondered if this construction was in the literature for a long time. I recently ran across it in "The Realm of Ordinal Analysis" by Michael Rathjen (proposition 2.17 on page 14). He couldn't find it in the rest of the literature, but the methods are sufficiently well known (and sufficiently close to Rosser's) to make it folklore. But, as emphasized by Rathjen, the result is significantly stronger than the usual theorems.

1.7 TWEEDLE-N

To push this further into uncharted territory, consider the infinite sequence of programs TWEEDLE-N (where N is an integer)

TWEEDLE-N:

1. Loop over M, printing the code of TWEEDLE-M into a variable $R(M)$.
2. Deduce the consequences of S, looking for a theorem of the form "TWEEDLE-M$\downarrow$" for some M.
3. If found, and $M = N$, it halts;
4. If found, and $M \neq N$, go into an infinite loop.

It is easy to see that either all TWEEDLE-N's run, or exactly one of them halts, something which S can prove, because steps 1+2 (which must be run simultaneously in two threads) are the same for all the programs. But S cannot prove that any single one of them runs.

To prove this, note that if there is an effective list of programs A_N (like the TWEEDLE's), you can make a program MERGE(A_N) which generates and runs all of the programs on parallel threads and halts exactly when any one of them does. Then S proves that either (TWEEDLE-k)$\downarrow$ or MERGE(A_r)$\downarrow$ (with $r \neq k$). That is, TWEEDLE-k and MERGE(all the others) form a DEE/DUM pair. This means that it cannot prove that one runs implies the other runs.

The result is that for any computable partition of the TWEEDLE's into two disjoint subsets A and B, S cannot prove that the TWEEDLE-A's run implies the TWEEDLE-B's run, although consis(S) proves that all the TWEEDLE's run. The theories "S+all the TWEEDLE-A's$\downarrow$" are sound theories, strictly between S and S+consis(S) in terms of Π^0_1 content – they prove new correct theorems about the non-halting of computer programs, but they are weaker than S+consis(S) (and weaker than each other in a way described by the partial order of set containment).

I like this theorem, because it is a proof which is very dastardly to translate to more traditional logic language. I think that computational language is more natural for these results.

I could go on making more complicated self-referential proofs (and I think this is an interesting thing to do, they all prove somewhat different things), but I will stop here to consider non self-referential proofs, which work at a higher level of the arithmetic hierarchy.

Chapter 2

TYPE II

These prove that there exist total functions which are not provably total. The statements in this case are Π_2^0, statements about the totality of some computable function.

2.1 FASTER-GROWTH

Given axiomatic system S, consider all computable functions f from the integers to the integers that are proven to be total (that is, which halt for all arguments). Now construct the program FASTER-GROWTH(n) which does the following:

- Lists the first n functions which are provably total, and computes their value at position n.
- Returns the biggest value at n, plus 1.

If S is sound for Π_2^0 statements, then there are infinitely many provably total functions, and FASTER-GROWTH halts at every input. Further, FASTER-GROWTH is eventually bigger than any function provably total in S. So "FASTER-GROWTH is total" is an unprovable true theorem.

The function FASTER-GROWTH is constructed entirely from other functions which are not equal to itself. The requirement on the theory is that when it proves a function is total, it is telling the truth, otherwise FASTER-GROWTH will get stuck in an infinite loop at some point. This is the Π_2^0 soundness. The Π_2^0 soundness proofs generally construct this type of thing, when unpacked.

The most common abbreviated form of this argument runs as follows: given an axiomatic system S, diagonalizing against all provably total recursive functions in the theory gives a total recursive function which the theory cannot prove is total. This argument is folklore.

Type II Gödel theorems provide a different way to strengthen the axiom system, by adding the statement of the totality of FASTER-GROWTH. This statement implies consistency of S, but is strictly stronger, since consistency is not enough to ensure FASTER-GROWTH is total (you need some soundness).

[About the same thing... from my annotations

> Fix some formal system $\mathcal{F}$. A total function is specified by the description of some TM M.
> For some Turing machines M there is a proof in $\mathcal{F}$ that $M(x)$ is total, that is, that the computation $M(x)$ halts for every x[1]. In $\mathcal{F}$ the (infinite) list of correct proofs of totality can be effectively enumerated as follows.

> Turing machine E:
>
> - For every string $x = \varepsilon$, 0, 1, 00, 01 ...:
>   - If x is a correct proof that some TM M defines a total function[a], then list M; to avoid repetitions, add "if not already listed".
>
> [a]This includes: (i) the checking of the candidate proof (checking proofs is recursive), (ii) the analysis of the last line of the proof (it should have the form "$\forall x : M(x)$ halts"), and (iii) the extraction of M from the last line.

> Thus, the following total Turing machine M' exists

> Turing machine M', input n:
>
> 1. Run E and get the n'th TM enumerated by E, call it M.
> 2. Run $M(n)$, which is total (if $\mathcal{F}$ is sound), and outputs $M(n) + 1$.

> It is assumed that $\mathcal{F}$ is sound, that is, that only true propositions are proved. Of course as (usually) there are true and false propositions, a sound formal system is necessarily consistent.
> The conclusion is that, for every sound formal system $\mathcal{F}$ there are total functions whose totality can not be proved in F.
>
> A similar argument is used to prove that the set of total (recursive) functions is not recursively enumerable. But here we are using Turing machines to enumerate formal proofs and prove the existence of unprovable true statements.]

[1]The number of such proofs is infinite, and in fact if there is a proof for the totality of M, there are infinitely many such proofs.

Chapter 3

TYPE III

Nonconstructive theorem about a large class of statements, which do not provide an explicit unprovable statement, and so cannot be used to step up the hierarchy of systems.

3.1 BOOLOS

There is no computer program which will output the true answer to statements of the form "Integer N can be named using k bytes or less worth of symbols of Peano Arithmetic"

Write program BOOLOS:

1. Loop over all integers N, looking for the first N which requires more than M symbols to name, where M is the length of the symbols describing the output of BOOLOS, translated to arithmetic.
2. Print N.

The contradiction means that BOOLOS does not work. Boolos is not so great, because it isn't focused on a particular system, but it's the same basic idea as "Chaitin", see below.

3.2 CHAITIN

Which replaces the notion of definability with Kolmogorov complexity, which is definability by an algorithm. Write the program CHAITIN to do the following:

1. List all proofs of S, looking for "the Kolmogorov complexity of string Q is greater than N" (where N is the length of CHAITIN).

2. Print string Q.

Now if S ever proves that the Kolmogorov complexity of any string is greater than the length of CHAITIN, then CHAITIN will make S into a Σ_1^0 liar (inconsistent). This proves that there is a completely effective bound on the maximum provable Kolmogorov complexity of any string (this was given previously as an answer).

There is an infinite list of true sentences of the form "The Kolmogorov complexity of Q is N", since there are infinitely many strings and only finitely many programs of length less than N. But only a finite number of these theorems get decided by any given axiom system S. This is a less explicit proof, because you can't be sure which strings are unprovably complex, so there isn't a natural axiom to add on to strengthen the system.

The statement "the Kolmogorov complexity of Q is N", translated to Arithmetic, is

$$\forall P\, \exists N : ((F^N(P) \text{ is a halted state with output } Q) \Rightarrow |P| > n)$$

so that it's Π_2^0.

Now to identify which proofs is what type:

- Self-referential sentence proofs – type I (Gödel, Rosser, Kleene, Post, Church, Turing, Smullyan, popular works).
- ε_0 induction proofs – these are type II, but specific to Peano Arithmetic. The general version is the one presented above (Kripke's proof, and Paris-Harrington, Goodstein, Hydra). The version they give is that the limit ordinal of all recursive provable ordinals is recursive but not provably recursive, but this is a type II argument.
- Jech/Woodin Set theory model proof – despite all its elegance and generality, the proof is type 1 when formulated computationally. I will elaborate below.
- Chaitin/Boolos – type III. I don't know any other type IIIs.

By the way, I agree with Sergei that finite axiomatizability (although emphasized by Putnam for some reason) is not so important. That property depends on exactly how you choose your axioms. The completeness theorem is strong enough to get a general computation from only finitely many axioms. The proof of impossibility of finite axiomatization (when it holds) is that the theory is self-reflecting, it can prove the consistency of any finite fragment (this is true in PA, because PA proves the consistency of induction restricted to level N in

the Arithmetic Hierarchy), and the axiomatization is weak, in that finitely many axioms are stuck in some finite fragment no matter how many times you use them. Self-reflection is interesting, but not that relevant for the incompleteness theorem.

To see that the Jech/Woodin proof is really a type I proof in disguise, it is important for the purpose of unpacking the construction to supplement the set theory with an effective procedure to give computational meaning to the models. This is just first order logic and the completeness theorem, as Andreas Caicedo States in the introduction.

Chapter 4

From Wikipedia article "Undecidable numbers"

Relationship with Gödel's incompleteness theorem

The concepts raised by Gödel's incompleteness theorems are very similar to those raised by the halting problem, and the proofs are quite similar. In fact, a weaker form of the First Incompleteness Theorem is an easy consequence of the undecidability of the halting problem. This weaker form differs from the standard statement of the incompleteness theorem by asserting that a complete, consistent and sound axiomatization of all statements about natural numbers is unachievable. The "sound" part is the weakening: it means that we require the axiomatic system in question to prove only true statements about natural numbers.

It is important to observe that the statement of the standard form of Gödel's First Incompleteness Theorem is completely unconcerned with the question of truth, but only concerns the issue of whether it can be proven.

Theorem 1 (Incompleteness from undecidability) *A complete, consistent, and sound axiomatization of all statements about natural numbers does not exist.*

Proof. Assume that we have a consistent and complete axiomatization of all true first-order logic statements about natural numbers. Then we can build a Turing machine that enumerates all these statements. This means that there is an Turing machine $N(n)$ that, given a natural number n, computes a true first-order logic statement about natural numbers such that, for all the true statements, there is at least one n such that $N(n)$ yields that statement.

With that assumption, we can decide if the Turing machine with representation a halts on input i as follows. This statement can be expressed with a first-order logic statement, say $H(a, i)$. Since the axiomatization is complete, it follows that either there is an n such that $N(n) = H(a, i)$ or there is an n' such that $N(n') = \neg H(a, i)$. Thus, a Turing machine that iterates over all n until

finding either $H(a, i)$ or $\neg H(a, i)$, always halts. This means that this Turing machine decides the halting problem. Since we know that there cannot be such a Turing machine, it follows that the assumption that there is a consistent and complete axiomatization of all true first-order logic statements about natural numbers must be false. $\square$

Chapter 5

Some comments to the post

I still am having some trouble with the full computational interpretation of Jech/Woodin. The simpler consequences are easy enough to interpret as standard type I arguments, but there is one theorem which is completely different: there is no descending infinite sequence of models of set theory. I had a similar proof for the well-foundedness of the collection of theories stronger than PA under the ordering A is stronger than B when A proves the consistency of B. But this theorem has a more involved proof than type I arguments.

The Jech/Woodin proof has an important ancestor, due to Kreisel, who came up with the first model-theoretic proof of the second incompleteness theorem in the 1960's (see, e.g., `logika.umk.pl/llp/06/du.pdf`

There are a couple well known proofs of incompleteness based on properties of PA degrees. PA degrees have been studied extensively in recursion theory.

A PA degree is a Turing degree that can compute a complete extension of PA. Obviously, to prove the incompleteness theorem, it's enough to show that no PA degree can be recursive. If we had a consistent r.e. theory T extending PA that was not incomplete, then its completion would be recursive – to decide whether $T \models \phi$ or $T \models \neg\phi$, simply look for a proof of ϕ or of $\neg\phi$ from T, and because T is assumed to be complete, this process always terminates and is hence recursive.

One way to see that there are no recursive PA degrees is to observe that any PA degree can compute a nonstandard models of PA via compactness and a Henkin construction. Now apply Tennenbaum's theorem that there are no recursive nonstandard models of PA.

Another way to see that there are no recursive PA degrees is with *piOne* classes. Every PA degree can compute a path through each Π^0_1 class (this is one direction of the Scott basis theorem). To finish, note that one can construct Π^0_1 classes that do not contain any recursive elements. For instance, there are Π^0_1 classes

that contain only diagonally nonrecursive elements.

Besides the proofs already listed, one essentially different treatment which comes to mind is Gentzen's consistency proof of PA, which established that PA can prove the well-ordering of ordinal notations less than ε_0 but could not prove the well-ordering of a notation for ε_0, and that, in turn, the well-ordering of ε_0 would suffice to prove the consistency of PA. Characterizing the proof-theoretic ordinal of a theory yields incompleteness results by an essentially different (and arguably far deeper / more general) route to that of Gödel. answer

answered Oct 5 at 0:31

—

Does Gentzen's proof actually establish incompleteness, though? My understanding is that Gentzen proves (i) that PA proves induction along (notations for) well-orderings of all order types $< \varepsilon_0$, and (ii) that T+Ind(ε_0) proves the consistency of PA, where T is a small sub-theory of PA. From this, we can conclude that PA does not prove Ind(ε_0), but to do so we need Gödel's second incompleteness theorem. That is, Gentzen proved a new instance of incompleteness, but relies on already knowing some other incompleteness. Is this correct? If so, this isn't what I was asking. – Noah S Oct 5 at 19:57

—

. . .

I think it would depend on whether there was any non-Gödelian route of establishing that successive powers $\omega^{\omega^{\cdots}}$ require increasing levels of quantification in PA. If so it's straightforward that you couldn't get to ε_0 without infinitely long formulas. Unfortunately I do not know the answer to this – I'm still struggling to understand Gentzen on a truly intuitive level. (Vide my question here: mathoverflow.net/questions/138875/. . .) But it "feels" like such a proof ought to exist.