

# Re: Applying Godel

---

*Source:* <http://sci.tech-archive.net/Archive/sci.logic/2008-09/msg00071.html>

---

- *From:* george <[greeneg@xxxxxxxxxxxxxx](mailto:greeneg@xxxxxxxxxxxxxx)>
  - *Date:* Fri, 5 Sep 2008 15:02:43 -0700 (PDT)
- 

On Sep 5, 12:45 pm, slartibartfast <[tomokane2...@xxxxxxxx](mailto:tomokane2...@xxxxxxxx)> wrote:

Ok..but when you follow those instructions ...what does a Godel sentence actually look like...can you give me a specific example of 1 Godel sentence for 1 particular theory T, (say PA.)

Prof.Smith's response will surely be just to read his book. The short version (the statement itself is NOT short, unless you predefine a short logical calculus) is that you have to DEFINE A PROOF PREDICATE.

The proof predicate needs to be binary and primitive recursive. Godel's theorem is a result specifically about standard classical first-order logic. It winds up applying to other things as well only because those things subsume FOL. So you have to start by formalizing the notion of first-order proof, and formalizing it in T, specifically (even though there is in some sense only 1 notion of standard classical 1st-order proof, different T's will yield different formalizations of it). Suppose S is the way you have chosen to describe the rules of inference for standard classical first-order logic. Then you can define a binary predicate  $P(p,q)$  that is true for and only for those pairs-of-terms  $(p,q)$  of your theory T, where p encodes a proof, using the axioms from T and the rules of inference from S, of the sentence-from-T that is encoded by the term(also-from-T)q.

The point is, you have to encode Both sentences AND lists-of-sentences (from T) as TERMS from T. When T is PA, this means you have to encode things like " $s(s(0)) < s(s(s(0)))$ " as  $s(s(s(s(s(s(s(s(s(s(s(s(0))))))))))))))$ .

In English we would say, "The Godel number of " $2 < 3$ " is 16." In practice the real Godel-numbers are much bigger. But the point

## Re: Applying Godel

is,  
you have to encode sentences as terms in order for this to work.  
You can also define a proof (for this purpose) as a list of sentences  
in T  
where each sentence follows by ONE application  
of a rule-of-inference in S to some sentences before it in the list.  
Once you have done that, you can then define Godel-numbers encoding  
lists  
of sentences-from-T as terms-from-T. Then you have to define Godel  
numbers  
for predicates, including defined predicates, including your proof  
predicate.

So "what it looks like" MUST DEPEND on how you have defined these  
godel numberings. Even then, it will be completely unparseable --- it  
will just look like  
s(s(s(s(s(s(s(s(.....))))))))))  
ad nauseam.

So the point is, YOU DON'T EVEN WANT to know what it REALLY looks  
like.

In order for you to see its structure at all, WE MUST INTRODUCE TONS  
of conventional  
abbreviations and syntactic shortcuts. Like decimal notation, for  
starters.

Or, we could just introduce an operator, that we could put around  
strings, meaning  
"the godel number of" whatever sentence (or part of a sentence, or  
list of sentences)  
you "see" inside the operator, as an argument to it.

In other words, not only can you not always GET what you want,  
SOMETimes, you can't even WANT what you want.

.