MAU11602
Lecture 21
2026-03-18

Computability and arithmeticity, consequences

The definition of arithmetic sets and functions was independent of any encoding.
For a formal language we choose an encoding to identify lists of symbols with natural numbers.
This gave us a notion of arithmetic sets of lists of symbols and arithmetic functions from such lists to such lists.
That notion was relative to a particular choice of encoding.
But we saw last time that the choice of encoding doesn't really matter. Encoding is computable, so change of encoding is arithmetic, and the composition of arithmetic functions is arithmetic.
Now that we know that all computable functions are arithmetic you might wonder whether all arithmetic functions are computable.
The answer is no. If it were yes there would in principle be an algorithm for solving all problems in arithmetic.

Self-reference

This sentence is false.

Logic, whether zeroeth order or first order, is not designed to deal with sentences like this. It's only intended to cope with sentences which are true or false, and explicit self-reference breaks this.

Explicit self-reference is also problematic in programming. As soon as we introduced recursion we lost the guarantee that all well typed definable functions evaluate to a value for all values of their arguments.

Even without recursion, most programming languages are capable of a form of implicit recursion though.

Suppose I want a self printing program. Rather than choose a programming language, let's try to give English language instructions for producing those same instructions.

The following would be explicitly self-referential: *Copy these instructions.*

The following is only implicitly self-referential: *Write out the sentence "Write out the sentence, pausing before the first comma and inserting a quoted version of the same sentence, before continuing to the end.", pausing before the first comma and inserting a quoted version of the same sentence, before continuing to the end.*

Quines

Write out the sentence "Write out the sentence, pausing before the first comma and inserting a quoted version of the same sentence, before continuing to the end.", pausing before the first comma and inserting a quoted version of the same sentence, before continuing to the end. doesn't refer to itself, only to a part of itself from which the rest can be reconstructed.

It's easier in some languages than in others but you can do this in any sufficiently powerful programming language. They generally follow the same pattern as the English sentence.

Such programs were named *quines* by Hofstadter and have become a niche hobby. You can almost certainly find multiple examples in your favourite language online.

Gödel's daring plan

Most of what I talked about last week, about encoding formal languages within Peano arithmetic, is due to Kurt Gödel, though I borrowed some improvements from other people.

Sometime in the late 1920's Gödel attempted to prove that Peano arithmetic is inconsistent.

Had he succeeded, this would have left mathematicians with two very unpleasant options:

- abandon arithmetic, or
- replace classical logic with something much weaker.

Even intuitionist logic wouldn't be weak enough, although minimal logic would still be okay.

Since pretty much everything was, and still is, proved using classical logic we'd have to start over from near zero.

The reason he thought he might succeed and the reason he failed are both interesting, as is what he ended up proving instead.

The original plan

We saw last week how to analyse statements in any formal language within Peano arithmetic using an encoding.
Peano arithmetic is itself a formal language, so we can express properties of expressions in Peano arithmetic as statements in Peano arithmetic.
Given an expression E in Peano arithmetic you could, for example find a Boolean expression F in Peano arithmetic expressing the fact that E is of type Boolean and has no free variables.
Using the idea behind quines you could even arrange that E and F are *the same statement*!
You can do this even though Peano arithmetic doesn't allow explicit self-reference, because it does have a form of quoting, namely encoding, and the encoding function is arithmetic.
So Gödel set out to find a Boolean expression G with no free variables saying the the expression G is of Boolean type and has no free variables *and is false*!

Is truth arithmetic?

To make this work you need to show that the encodings of false statements form an arithmetic set, which is equivalent to showing that the encoding of true statements form an arithmetic set.
Can't we handle this the way we treated evaluation of programs, so true expressions are those expressions which evaluate to true?
Unfortunately, or maybe fortunately, no. Quantifiers mean our interpretation of Peano arithmetic isn't algorithmic.
For example

$$\neg\exists a.\forall b.\,(\forall c.\,(\exists d.b = c \cdot d) \rightarrow c = S0 \vee c = b) \rightarrow b \leq a$$

is true, but we can't find this out by testing all a and seeing that none work.
We can't even test any particular a because we'd need to test all b.
To test even a single b we'd need to test all c, or would we?

Bounded arithmetic

We were considering the expression

$$\neg\exists a.\forall b.\,(\forall c.\,(\exists d.b = c \cdot d) \to c = S0 \vee c = b) \to b \leq a$$

The subexpression

$$\forall c.\,(\exists d.b = c \cdot d) \to c = S0 \vee c = b$$

isn't directly something we can evaluate algorithmically for given a and b but the parenthesised part says c is a divisor of b and all divisors of b are less than or equal to b, so it would be equivalent to write

$$\forall c.c \leq b \to (\exists d.d \leq b \wedge b = c \cdot d) \to c = S0 \vee c = b.$$

To check this statement for a given a and b we'd only have to check finitely many c and d, those up to b.

Bounded arithmetic, continued

It's customary to abbreviate

$$\forall c.c \leq b \to (\exists d.d \leq b \wedge b = c \cdot d) \to c = S0 \vee c = b$$

to

$$\forall c \leq b.(\exists d \leq b.b = c \cdot d) \to c = S0 \vee c = b.$$

We could consider a language like that of Peano arithmetic but which only allows bounded quantifiers, i.e. $\forall X \leq A.P$ or $\exists X \leq A.P$, where X is a variable, A is an expression of type natural, and P is an expression of type Boolean.
Free occurences of X in P are to be considered bound in the whole expression but free occurences of X in A are not.
There is an algorithm for evaluating expressions in bounded arithmetic, and this can be turned into a proof that encodings of Boolean expressions in bounded arithmetic with no free variables which evaluate to true, or those that evaluate to false, form an arithmetic set.

The tower

Arithmetic sets were defined by the existence of an expression in Peano arithmetic expressing set membership. We could define bounded arithmetic sets similarly.
The intended interpretation of bounded arithmetic sets is (bounded arithmetic) sets, not bounded (arithmetic sets).
English could really use parentheses.
Is the set of true statements in bounded arithmetic bounded arithmetic?
Is encoding a bounded arithmetic function?
If so then we can realise Gödel's dream with bounded arithmetic.
I wasn't being careful with quantifiers, but even if you are the answer is still no.
There's a tower of languages, with bounded arithmetic at the bottom and each floor containing expressions obtained by unbounded quantification of expressions from the floor below and the other allowed operations (including bounded quantification).
Peano arithmetic is the full tower, with all floors.
The tower is infinite, but each expression lives on some finite floor.

The other tower

There's also a tower of sets, with the bounded arithmetic sets as the ground floor, and the $n+1$'st floor containing those sets which are definable using expressions on the $n+1$'st floor of the tower of expressions, but not using any on the n'th floor.
It may not be obvious which floor a set belongs to. The set of primes looked like it belonged to the second, but is actually on the ground floor.
It's not even obvious if this tower is finite.
If truth in Peano arithmetic is arithmetic then in fact the tower is finite, because once we reach its floor we can use it to replace any further unbounded quantifiers.
Then we can finally fulfill Gödel's dream of destroying mathematics once and for all!

Tarski, semantic incompleteness

There's another point of view on all of this. If we assume arithmetic is consistent then the strategy on the previous slide can't work, so we can prove that the set of encodings of true statements of Peano arithmetic is not arithmetic.
Actually we can prove it even if arithmetic is inconsistent, since from a contradiction we can prove anything we want.
So the non-arithmeticity of truth in Peano arithmetic is a theorem: Tarski's theorem.
Suppose we take a different approach, and use proofs as evidence of correctness of statements.
With suitable axioms and rules of inference proof verification is arithmetic, so provability is arithmetic.
In other words, the set of encoding of theorems of suitably axiomatised Peano arithmetic is arithmetic.
So thee set of encodings of true statements can't be the same as the set of encodings of theorems, and the set of true statements and the set of theorems are not the same.
Arithmetic must be either inconsistent or semantically incomplete.