24.118 – Paradox and Infinity
Problem Set 9: Gödel's Theorem

1 A Super-Short Proof of Gödel's Theorem

You are all familiar with arithmetic: the study of the natural numbers (Zero, One, Two, Three, etc..), and its two fundamental operations, addition and multiplication.

How complex is the set of arithmetical truths? Is it in principle possible to program a computer so that it outputs, one by one, all the truths of arithmetic—and never outputs a falsehood? (Since there are infinitely many arithmetical truths, our computer will never complete its task—it will go an working forever—but that's okay: it is enough for present purposes if every mathematical truth is listed at some point.)

If we were interested in this project, the first step would be to decide what language our computer should use in announcing its results. Let us suppose that the relevant language contains the symbols '0' and '1' (which name Zero and One, respectively), and the symbols '$+$' and '$\times$' (which express addition and multiplication, respectively). This is enough to name all natural numbers. (The number Two, for example, can be named by the expression '$1+1$', which we shall abbreviate '2'. And similarly for other numbers.) Let us also suppose that our language contains the parenthesis-symbols '(' and ')', and the identity-symbol '$=$' (which expresses identity between numbers). Our language will then be able to formulate simple mathematical truths such as '$2 \times 1 = 1 \times 2$', or '$2 \times (2+2) = (2 \times 2) + (2 \times 2)$'.

So far, however, we are unable to express *generality.* We are unable express, for example, the fact that, for *any* numbers a and b, the product of a and b equals the product of b and a. Let us therefore introduce the symbol '$\forall$', (which means 'for any'), and *variables* such as 'a' and 'b'. We can now express the commutativity of multiplication as '$\forall a \forall b(a \times b = b \times a)$' (i.e. 'for any number a and any number b, a times b equals b times a.')

Finally, we introduce the symbols '$\neg$' and '$\wedge$' to express negation and conjunction, respectively. This allows us to express statements like '$\neg(1=2)$' (i.e. '*it is not the case that* one equals two') and '$\forall a((a+0=a) \wedge (a \times 0 = 0))$' (i.e. 'for every number a, a plus Zero equals a, *and* a multiplied by Zero equals Zero').

Despite being so incredibly simple, the language we have built is extraordinarily powerful. It can be used to express *existential* statements. (For example, the sentence 'there is a number that results from dividing 12 by 4' can be expressed as '$\neg\forall a \neg(4 \times a = 12)$'.) It can also be used to express *conditional* statements. (For example, the sentence 'if the result of multiplying a number by itself is One, then that number must be One' can be expressed as '$\forall a \neg(a \times a = 1 \wedge \neg(a = 1))$'.)

And this is only the beginning. Our language is able to capture large numbers of interesting arithmetical propositions. It can express, for example, 'there are infinitely many prime numbers', and 'there are no numbers a, b and c such that $a^3 + b^3 = c^3$'. In fact, our language is *so* powerful that it can express *all* of the key results of arithmetic.

It can describe, moreover, the operation of certain computer programs. It contains, for example, a formula '$\phi(n, m)$' which is true if and only if the nth C++ program (according to some canonical ordering) halts having printed a sequence of exactly m ones.

The expressive richness of our language guarantees that our computer-building project is truly interesting. If we could program a computer so as to list all the truths that our language is capable of expressing (and no falsehood), we would have succeeded in concentrating a huge wealth of mathematical knowledge in a finite list of lines of code. Implicit in our program would be not just all the arithmetic we learned in school but also the solution to difficult mathematical problems, such as Fermat's Last Theorem, and answers to unsolved mathematical problems, such as Goldbach's Conjecture.

As it turns out, however, it is *logically impossible* to construct such a computer. In 1931, the great Austrian mathematician Kurt Gödel proved that the only way to get a computer to list all mathematical truths expressible in our language is for the list to include some falsehood.

Forty years later, the Argentine-American mathematician Gregory Chaitin discovered a new method for proving Gödel's result.[1] And, as we shall see, Chaitin's proof is surprisingly simple.

The fundamental concept we will be using in the proof is that of the *Kolmogorov Complexity* of a natural number. Say that a computer program *generates* the number n if it outputs a sequence of exactly n ones, and halts. The Kolmogorov Complexity of a number (relative to a give programming language; as it might be, C++) is the number of symbols in the shortest program (of C++) that generates n.

As it turns out, the language we described above can express a certain formula '$K(n) = m$' with the property that, for all n and m, '$K(n) = m$' is true if and only if the Kolmogorov Complexity of n is m.

Let L be a large enough natural number. (It will become clear later what 'large enough' means.) There must be some number n such that the sentence '$K(n) > L$' is true. (Why? That's problem 4 below.) Therefore, if we were able to build a computer program, P, that outputs all arithmetical truths (and no falsehood), P must output at least one statement of the form '$K(n) > L$'.

Let z be the natural number such that '$K(z) > L$' is the first statement of the form '$K(n) > L$' outputted by P. The Kolmogorov Complexity of z cannot be much greater than the number of symbols in P. The reason is simple. Suppose we want to write a computer program that generates z. We can do so by introducing a small change in P. Instead of asking our program to output mathematical truths as they are identified, we ask it to work in silence until it identifies a statement of the form '$K(n) > L$'. At that point, the program is asked to output a sequence of exactly n ones, and halt. Since the first

[1]Chaitin's Proof is described in a recent article by Shira Kritchman and Ran Raz, along with a related proof of Gödel's Second Incompleteness Theorem: "The Surprise Examination Paradox and the Second Incompleteness Theorem", *Notices of the AMS* , 57 (11), available at:

http://www.ams.org/notices/201011/rtx101101454p.pdf statement of the form '$K(n) > L$' that P identifies is '$K(z) > L$', the modified program will output a sequence of exactly z ones, and halt.

In order to modify P in this way, we would have to add some extra lines of code, but not many. This is all we need to do: (1) we need to give our program the number L, and (2) we need to ask our program to work in silence until it identifies a statement of the form '$K(n) > L$', at which point in should output a sequence of exactly n ones, and halt. The first of these tasks will require, at most, of $log(L)$ symbols. (That is because, in general, a number m can be represented as a sequence of $log(m)$ digits.) The second of task will require a small number of additional symbols, which will by represented by a constant c, which does not depend on the size of L.

Our modified program will therefore consist of no more than $log(L) + c + |P|$ symbols, where $|P|$ is the number of symbols in P.

Since the modified program generates z, we know that $log(L) + c + |P| \geq K(z)$. But if L is large enough, it must also be the case that $L > log(L) + c + |P|$ (because c and $|P|$ are constant, and natural numbers grow much faster than their logarithms). From these two inequalities it follows that $L > K(z)$, and therefore that the statement '$K(z) > L$' must be false.

This is terrible news. We had earlier assumed that P outputs '$K(z) > L$' as part of its effort to list all mathematical truths (and no falsehood). But if it turns out that '$K(z) > L$' is false, as we have concluded, our program will have outputted a falsehood.

What has happened, in short, is the following. First we showed that in order for P to output every true statement expressible in our language, it would have to output some statement of the form '$K(n) > L$'. We then showed that the first statement of this form that P outputs must be false. Our project is therefore hopeless: it is impossible to write a program that lists all and only true statements of our arithmetical language.

2 Problems

You will be graded both on the basis of whether your answers are correct and on the basis of whether they are properly justified. There is no word limit.

1. I claim in the text that the statement 'if the result of multiplying a number by itself is One, then that number must be One' can be expressed as '$\forall a \neg(a \times a = 1 \wedge \neg(a = 1))$'. Show that, in general, 'if p then q' can be paraphrased as '$\neg(p \wedge \neg q)$'. (If you know about truth-tables, use truth-tables; if not, use an informal argument.) (5 points)

2. I claim in the text that our language can express 'there are infinitely many prime numbers', and 'there are no numbers a, b and c such that $a^3 + b^3 = c^3$'. Show that this is right, by exhibiting the relevant sentences. (Note that you are *not* being asked to prove the relevant statements; all you need to do is show is that the statements can be *expressed* in a particular language by exhibiting some sentences of that language.) (5 points)

3. I introduce L by stipulating that it is to be 'large enough', and promise that it will later become clear what large enough means. What does 'large enough' mean? (5 points)

4. I claim in the text that there must be some number n such that the sentence '$K(n) > L$' is true. Show that this is so. (5 points)

5. *Extra Credit:* For a Turing machine to compute the (total) function $f(n)$ from the natural numbers to the natural numbers is for it to output a string of $f(n)$ ones, and halt, when given a sequence of n ones as input. For a (total) function from the natural numbers to the natural numbers to be Turing Computable is for it to be computed by some Turing machine.

   The proof in the main text contains a hidden proof of the claim that Kolmogorov Complexity is not Turing Computable. Find it. In other words, prove that Kolmogorov Complexity is not Turing computable by using the proof in the main text as a hint. (5 points)

MIT OpenCourseWare
http://ocw.mit.edu

24.118 Paradox & Infinity
Spring 2013

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.