24.118: Paradox and Infinity
PS 4: Suggested solutions

November 8, 2001

(4 pts each)

Problem 1

Suppose the partial function f is calculable. Then there is an algorithm F which, when given input n with $n \in \text{Domain}(f)$, outputs $f(n)$ after finitely many steps. To show that S=$\{\text{Pair}(m,n)|m \in \text{Domain}(f) \wedge f(m) = n\}$ is effectively enumerable, we will find an algorithm that lists all the members of S.

The idea is to use algorithm F to calculate, for each number n, the values of $f(n)$ and then list $\text{Pair}(n, f(n))$. The only difficulty is that f is *partial*, so the algorithm F may not give an output for every n and for some inputs n it may just loop forever. In setting up our algorithm for listing S, we must make sure that our listing of the members of S doesn't stop prematurely just because F got stuck on some n we fed it. The trick is to work on many computations $(f(0), f(1), f(2), \ldots, f(n))$ simultaneously, doing a step at a time. This is how:

Abbreviation: Call the computation of F with input n '$F(n)$'

We start by doing the first step of the calculation of $F(0)$. If it gives output n, then calculate $\text{Pair}(0, n)$ and list it (note that the Pair-function is calculable). Next, start calculating $f(1)$ using algorithm F. After one step of running $F(1)$, stop $F(1)$. If $F(1)$ returned output n, then calculate $\text{Pair}(1, n)$ and list it. Now go back to calculation $F(0)$. If the last time we visited it, it hadn't yet given an output, do a further step of the computation. If $F(0)$ returns n after that step, then calculate $\text{Pair}(0, n)$ and list it. Go back to calculation $F(1)$. If the last time we visited it, it hadn't yet given an output,

do a further step of the computation. If $F(1)$ returns m after that step, then calculate $\text{Pair}(1, m)$ and list it. Start calculating $f(2)$ using algorithm F. After one step of running $F(2)$, stop $F(2)$. If $F(2)$ returned output m, then calculate $\text{Pair}(2, m)$ and list it. Next, go back to $F(0)$ ...
In general, we go along the following pattern to weave together the various computations (this process is sometimes called 'dovetailing' and is fairly widely used in computability theory):

$F(0)$	$F(1)$	$F(2)$	$F(3)$	$F(4)$	$F(5)\ldots$
1	2	5	9	14	...
3	4	8	13	...	
6	7	12	...		
10	11	...			
15	......				

Each time we visit a calculation $F(n)$, we do another step of it, if it returns a value (which is going to be the $f(n)$ we were looking for) after that step we calculate $\text{Pair}(n, f(n))$ and list it. If it doesn't return a value (yet), we go on to work a bit on some of the other computation and come back to it later. This way we make sure that for all n, if n is in the domain of f we find $f(n)$ and list $\text{Pair}(n, f(n))$. This algorithm lists all the members of S. So S is effectively enumerable.

Conversely, suppose S is effectively enumerable. We need to find an algorithm which, when given n as input, returns $f(n)$ as output if $n \in \text{Domain}(f)$ and doesn't return anything if $n \notin \text{Domain}(f)$.
To calculate $f(n)$ do the following: Start listing S. Check the first element i of the listing. If i codes a pair of numbers whose first element is n, i.e. $i = \text{Pair}(n, x)$ for some x, then output x. Otherwise look at the next element of of the listing of S and repeat the checking procedure. Continue until you find an element i in the listing with $i = \text{Pair}(n, x)$ for some x.
Since for every $n \in \text{Domain}(f)$, S contains $\text{Pair}(n, x)$ where $x = f(n)$, this procedure is guaranteed to output a value of $F(n)$. Thus, f is calculable.

Problem 2

Suppose the total function f is calculable. To show that $S = \{\text{Pair}(m, n) | m \in \text{Domain}(f) \wedge f(m) = n\}$ is decidable we will show that there is an algorithm

such that if we input a number x, it will output 1 if x codes a pair $\langle n, m\rangle$ with $f(n) = m$ and it will output 0 otherwise.
Given input x, do the following: Find n, m such that $\text{Pair}(n, m) = x$.[1] Then calculate $f(n)$ using the fact that f is calculable. If $f(n) = m$, then output 1, otherwise output 0.
This constitutes a decision procedure for S, so S is decidable.

Conversely, suppose S is decidable. To calculate $f(n)$ do the following: Run the decision procedure D for S with input $\text{Pair}(n, 0)$. If D returns 1, output 0. Otherwise, run D with input $\text{Pair}(n, 1)$. If D returns 1, output 0. Otherwise run D with input $\text{Pair}(n, 2)\ldots$ run D with input $\text{Pair}(n, m)$. If D returns 1, output m...
Since D is total, and S contains codes for all the argument-value pairs for f, this will find the value for $f(n)$ for every n. Thus, f is calculable.

Problem 3

Let D be the set of sets S such that either S or its complement is effectively enumerable. Since there are only countably many algorithms, D can only be countable. That means that not all sets of natural numbers (of which there are uncountably many) can be in D.

Here is a more *constructive* argument: Consider the set T of numbers that codes all and only the true sentences of PA. This set is not effectively enumerable and neither is its complement.
This is why: If T were effectively enumerable, then arithmetic (all the true sentences in the language of PA) would be axiomatizable (ie there would be an effective listing of a collection of sentences from which *all* the truths of arithmetic can be derived). But we have seen that arithmetic is not axiomatizable (Gödel). Similarly, if T's complement TC were effectively enumerable, arithmetic would be axiomatizable (just list TC and for every element listed, if it codes a sentence of arithmetic, list the negation).

4.

Suppose S is effectively enumerable. That means that there's an algorithm, A which lists the members of S. Note that when starting to run A, we

[1] Again, there is an algorithm to do this. We will not show this here but just assume it.

do not yet know whether or not it will list any numbers and if it does list numbers, whether it lists finiteley or infinitely many.
We need to define a *total*, *calculable* function f such that S is the range of f.

Define f as follows:

$f(0) =$ the first element listed by A.

$f(n) =$ the element listed by A at the nth step of the computation of A *after* it listed the first element, if such an element exists (for $n > 0$)

$f(n) = f(n-1)$ otherwise (for $n > 0$)

There are two cases: *Either* A does not list any numbers. Then f is undefined for all n and hence not a *total* calculable function. But in this case S is empty, so the claim holds. *Or* A does list numbers. In this case, S is the range of f, f is total and f is calculable.

Many of you defined a function g by letting $g(n)$ be the nth element of the listing. the problem with that is that A may only list *finitely many* numbers (because S is finite). In this case, the simple definition, $g(n) =$ the nth element listed, won't work. You need a *total* function, so you must make sure that g(n) has a value even for n bigger than the cardinality of S. The above definition takes care of this.

Conversely: If S is empty, then S is effectively enumerable. Suppose S is the range of the total calculable function f. Then effectively list S as follows: At the nth state of the listing, compute $f(n)$ and list it. Since f is total, this will work.

5.

If S is the range of a 1-1 calculable total function, S is effectively enumerable by the argument in 4. If S is finite, S is effectively enumerable (an algorithm to list it may use a lookup table with all the members of S. Since there are only finitely many, such a lookup-table can be part of an algorithm)

Conversely: If S is effectively enumerable, and F is an algorithm which lists S, define f as follows:

$f(0) =$ the first element e listed by F

$f(n)$ = the first element e listed by F such that for all $0 < i < n$: $e \neq f(i)$ (for $n \neq 0$)

There are 3 cases: (i) S is empty, then f is undefined for all n. But S is finite and so the claim holds. (ii) S is finite and not empty (then f is not definined for all but finitely many n), in which case we are done. (iii) S is infinite. Then f is 1-1, total and calculable and S is the range of f.

6.

There are two cases:

1. There are only finitely many n such that there is a sequence of n or more consecutive '7's in the decimal expansion of π.
2. For every n, there is a sequence of n or more consecutive '7's in the decimal expansion of π.

In the first case, h, considered as a set,[2] looks as follows: There are finitely many pairs whose second member indicates a place where a sequence of 7s begins: $\langle 0,0\rangle, \langle 1,m_1\rangle, \langle 2,m_2\rangle, \langle 3,m_3\rangle, \langle 4,m_4\rangle \ldots, \langle n,m_n\rangle$, the remaining members of h are all of the form $\langle n,0\rangle$. But this set is decidable: Just build a lookup table with the (finitely many) pairs $\langle n,m\rangle$ where $n,m \neq 0$. Then to calculate $h(n)$, check the table to see whether $\langle n,m\rangle$ appears in it for some m. If so, output m, if not output 0.

In the second case, determine $h(n)$ as follows: Start listing the decimal expansion of π. Keeping track of your position as you move through the list, search until you find the first sequence of at least n consecutive '7's. Output the position at which it starts.
In either case, h is calculable. We just don't know which case applies.[3]

[2] Recall: A function is a set S of pairs $\langle n,m\rangle$ such that if $\langle n,m_1\rangle \in S$ and $\langle n,m_2\rangle \in S$, then $m_1 = m_2$; see handout 1

[3] Note: This works because $h(n)$ is defined as the place at which a sequence of n *or more* '7's starts. If instead we had a function [h' with $h'(n)$ = the place in the decimap expansion of ϕ at which a sequence of *exactly* n '7's starts] then there would be another case: In h, there could be *infinitely many* pairs $\langle n,m\rangle$ without there being such a pair for every n. It's interesting to see how such a small difference in how a function is definied can make such a huge difference to whether or not its calculable.

7.

Note that in principle it is straightforward to extend a partial function to a total function. If f is partial, just define g as follows:

$g(n) = f(n)$ if $n \in \text{Domain}(f)$

$g(n) = 1$ otherwise

Then g extends f (ie g has the same values as f for all numbers in the domain of f) and g is total.

It is less straightforward to extend a partial *computable* function f to a total *computable* function g. The potential difficulty lies in finding out for what numbers f is not defined (these are the numbers for which we cannot let $g(n) = f(n)$). If there is an effective way of telling, for each n, whether n is in the domain of f (ie if the domain of f is decidable), this is easy: In this case, define g as above—since the domain of f is decidable, the resulting function will be calculable. Many of you gave this type of example. As you can see, this doesn't work.

To find a partial calculable function f that cannot be extended to a total computable function g, we need a function whose domain is not decidable. An example is the following:

$f(n) = m$ if the nth program with input n outputs m

This is partial, because not every program halts and gives an output when run with some given input. It is calculable, because to calculate $f(n)$ for n in the domain of f, we run the nth program with input n, and if it gives an output, we return it as result.
However, f cannot be extended to a total calculable function because there is no algorithm to tell, for arbitrary n, whether the nth program with input n halts. (The Halting problem is unsolvable)

Note, however, that there are partial calculable functions with undecidable domains which *can* be extended to total calculable functions. For instance, suppose the domain of calculable function f is undecidable, but for all n, if n is in the domain of f, then $f(n) = 1$. In this case f can easily be extended to a total calculable function: Define $g(n) = 1$ for *all* n.