Math 55 - Spring 2004 - Lecture notes # 4 - Jan 29 (Thursday)

Goal: Use (roughly) the same proof to show that
1) There are more real numbers than rational numbers
2) There are more functions with domain N than programs,
(so that not every function can be implemented by a program)
3) A "super-debugger" or ""infinite loop finder" is a
particular function that cannot be implemented.

Rough plan: Use proof by contradiction, i.e. assume the opposite
is true (eg there are the same number of reals as rationals,
or functions as programs) and get a contradiction.
More precisely, we will assume that the set of reals (or functions)
is countable, and get a contradiction.

Recall Def: A set is countable if either it is finite, or
can be put in one-to-one correspondence with N = {1,2,3,...}

Recall last time: Q = {rationals} is countable
J = {syntactically correct programs} is countable

1) Theorem 1: The real numbers are not countable.
Proof: It is enough to show that the set of
reals between 0 and 1
is not countable, because the set of all reals is even bigger.
We assume that these real numbers are countable. This
means that there exists a list r(1),r(2),r(3),... in which each
real number appears exactly once.
We will get a contradiction by constructing another real number s
that can't be in this list.
Now write each real number as a decimal fraction, eg
r(1) = 1/2 = .500...
r(2) = 1/3 = .3333...
r(3) = pi - 3 = .141592...
r(4) = ...
When there are two ways to write the same number
(eg 1/2 = .5000... = .49999...) then choose the one that ends in 0s.

To construct a real number s that is not on this list,
we will write down a decimal fraction that
can't be r(1) because we choose s's 1st digit different from r(1)'s
can't be r(2) because we choose s's 2nd digit different from r(2)'s

```
   can't be r(3) because we choose s's 3rd digit different from r(3)'s
   ...
   can't be r(i) because we choose s's ith digit different from r(i)'s
   ...
   and also so that it doesn't end in nines.
   There are many ways to do this, for example by the rule:
     "If the i-th digit of r(i) is 1, let the i-th digit of s be 2.
      Otherwise, if the i-th digit of r(i) is not 1, let the i-th
      digit of s be 1."
   For example, for our list above, we set s = .112...
   Since this trick works for any proposed complete list r(1),r(2)...
   of the real numbers, no such complete list can exist.
   This completes the proof.

2) Theorem 2: the set of functions is not countable.

   Since the proof of Theorem 2 is almost the same as Theorem 1,
   we will CAPITALIZE the few differences between them below.

   Proof: It is enough to show that the set of
   FUNCTIONS WITH DOMAIN N AND CODOMAIN D = {0,1,2,3,4,5,6,7,8,9}
   is uncountable, because the set of all FUNCTIONS is even bigger.
   We assume that these FUNCTIONS are countable. This
   means that there exists a list r(1),r(2),r(3),... in which each
   FUNCTION appears exactly once.
   We will get a contradiction by constructing another FUNCTION s
   that can't be in this list.
   Now write each FUNCTION's OUTPUT as a LIST OF DECIMAL DIGITSs, eg
    r(1)(N)   = (500...)
    r(2)(N)   = (3333...)
    r(3)(N)   = (141592...)
    r(4)(N)   = ( ...
   (STRINGS OF NINES MAY APPEAR.)

   To construct a real number s that is not on this list,
   we will write down a LIST OF DECIMALS that
   can't be r(1)(N) because we choose s's 1st digit different from r(1)
   can't be r(2)(N) because we choose s's 2nd digit different from r(2)
   can't be r(3)(N) because we choose s's 3rd digit different from r(3)
   ...
   can't be r(i)(N) because we choose s's ith digit different from r(i)
   ...
```

There are many ways to do this, for example by the rule:
"If the i-th digit of r(i)(N) is 1, let the i-th digit of s be 2.
Otherwise, if the i-th digit of r(i) is not 1, let the i-th
digit of s be 1."
For example, for our list above, we set s(N) = (112...)
Since this trick works for any proposed complete list r(1),r(2)...
of FUNCTIONS, no such complete list can exist.
This completes the proof.

The idea of building a "counterexample" (real not on the list,
function not on the list) by going down the diagonal of a table
(eg one row for each real, one column for each decimal place)
is called "diagonalization". We will use the same proof technique
for the next result.

3) To show that a "super-debugger" or "infinite loop finder" cannot
be implemented requires a little more notation.
A superdebugger is a program that implements the function
S(program P,input I) = [1 if P halts on input I
[0 if P goes into infinite loop on input I
In particular the program implementing S should never go into an
infinite loop itself. Another name for the question of
whether a "super-debugger" exists is the "halting problem"
(See also section 3.1 of the text, and the notes by W. Kahan
on the class web page.)

As before, we restrict ourselves to the case where the input
is a single integer, since if this can't be solved, then neither
can the general case. And since the set of all programs is
countable, we can write them in a list p(1), p(2),... and
so think of the function S as a function of two integers:
S(j,i) = [1 if program j halts on input integer i
[0 if program j goes into infinite loop on input i
To get a contradication, we
1) Assume that we have a program H(j,i) (for "halting") that can
indeed compute S(j,i) without itself going into an
infinite loop, and then
2) Construct another program C (for "contradiction") for which
H must give the wrong answer, using diagonalization.

Here is the idea of C in a table. In row j of the table,
we record whether p(j) halts or loops on each input

```
 by writing down a list of Os and 1s, where the i-th entry
 is 1 if p(j) halts on input i, and 0 if it loops.
 The table looks like this:
   p(1): 00110...
   p(2): 10101...
   p(3): 11110...
   ...
 Tor example, program p(3) halts on input i=4 because the
 4th digit in "11110..." is 1.
 The idea is to build a program C that that is not on this list,
 because C
can't be p(1) because we choose C to differ from p(1) on input 1
  (ie C loops if p(1) halts and C halts if p(1) loops)
can't be p(2) because we choose C to differ from p(2) on input 2
can't be p(3) because we choose C to differ from p(3) on input 3
...
 Here is C, which calls H as a subroutine:
    progam C(integer i)
       if H(i,i) = 1  ... program i halts on input i
          go into an infinite loop
       else           ... program i loops on input i
          stop
       endif
 C is constructed to differ from each program on the "diagonal"
 (i,i) of the table.

 Here is the contradiction. Since C is itself a program,
 it must appear on the list of programs, say as program number c.
 But it can't be on the list because its behavior
 (halting or looping) differs from each program in the table.

 In more detail, we get a contradiction by asking what
 happens when we try to compute C(c), i.e.
 run C on itself. The idea is that H(C,C) should tell us
 whether C(c) goes into an infinite loop or not.
 There are two cases:
  1) Suppose H(c,c)=1, that is C(c) is supposed to halt.
     But looking at the code for C, we see that
     C(c) goes into an infinite loop if H(c,c)=1, a contradiction.
  2) Suppose H(c,c)=0, that is C(c) is supposed to loop.
     But looking at the code for C, we see that
     C(c) halts if H(c,c)=0, a contradiction.
```

So assuming that the program H(j,i) existed led to a contradiction.
So it can't exist.