CS 70 Discrete Mathematics and Probability Theory
Spring 2017 Rao

DIS 6a

1 Count It!

For each of the following collections, determine and briefly explain whether it is finite, countably infinite (like the natural numbers), or uncountably infinite (like the reals):

(a) $\mathbb{N}$, the set of all natural numbers.

(b) $\mathbb{Z}$, the set of all integers.

(c) $\mathbb{Q}$, the set of all rational numbers.

(d) $\mathbb{R}$, the set of all real numbers.

(e) The integers which divide 8.

(f) The integers which 8 divides.

(g) The functions from $\mathbb{N}$ to $\mathbb{N}$.

(h) Computer programs that halt.

(i) Computer programs that always correctly tell if a program halts or not.

(j) Numbers that are the roots of nonzero polynomials with integer coefficients.

Solution:

(a) Countable and infinite. See Lecture Note 10.

(b) Countable and infinite. See Lecture Note 10.

(c) Countable and infinite. See Lecture Note 10.

(d) Uncountable. This can be proved using a diagonalization argument, as shown in class. See Lecture Note 10.

(e) Finite. They are $\{-8,-4,-2,-1,1,2,4,8\}$.

(f) Countably infinite. We know that there exists a bijective function $f:\mathbb{N}\to\mathbb{Z}$. Then function $g(n)=8f(n)$ is a bijective mapping from $\mathbb{N}$ to integers which 8 divides.

(g) Uncountably infinite. We use the Cantor's Diagonalization Proof:

Let $\mathscr{F}$ be the set of all functions from $\mathbb{N}$ to $\mathbb{N}$. We can represent a function $f \in \mathscr{F}$ as an infinite sequence $(f(0), f(1), \cdots)$, where the i-th element is $f(i)$. Suppose towards a contradiction that there is a bijection from $\mathbb{N}$ to $\mathscr{F}$:

$$\begin{aligned}
0 &\longleftrightarrow (f_0(0), f_0(1), f_0(2), f_0(3), \ldots) \\
1 &\longleftrightarrow (f_1(0), f_1(1), f_1(2), f_1(3), \ldots) \\
2 &\longleftrightarrow (f_2(0), f_2(1), f_2(2), f_2(3), \ldots) \\
3 &\longleftrightarrow (f_3(0), f_3(1), f_3(2), f_3(3), \ldots) \\
&\vdots
\end{aligned}$$

Consider the function $g : \mathbb{N} \to \mathbb{N}$ where $g(i) = f_i(i) + 1$ for $i \in \mathbb{N}$. We claim that the function g is not in our finite list of functions. Suppose for contradiction that it did, and that it was the n-th function $f_n(\cdot)$ in the list, i.e., $g(\cdot) = f_n(\cdot)$. However, $f_n(\cdot)$ and $g(\cdot)$ differ in the n-th number, i.e. $f_n(n) \neq g(n)$, because by our construction $g(n) = f_n(n) + 1$ (Contradiction!).

(h) Countably infinite. The total number of programs is countably infinite, since each can be viewed as a string of characters (so for example if we assume each character is one of the 256 possible values, then each program can be viewed as number in base 257, and we know these numbers are countably infinite). So the number of halting programs, which is a subset of all programs, can be either finite or countably infinite. But there are an infinite number of halting programs, for example for each number i the program that just prints i is different for each i. So the total number of halting programs is countably infinite.

(i) Finite. There is no such program because the halting problem cannot be solved.

(j) Countably infinite. Polynomials with integer coefficients themselves are countably infinite. So let us list all polynomials with integer coefficients as $P_1, P_2, \ldots$. We can label each root by a pair (i, j) which means take the polynomial P_i and take its j-th root (we can have an arbitrary ordering on the roots of each polynomial). This means that the roots of these polynomials can be mapped in an injective manner to $\mathbb{N} \times \mathbb{N}$ which we know is countably infinite. So this set is either finite or countably infinite. But every natural number n is in this set (it is the root of $x - n$). So this set is countably infinite.

2 Halting Problem Sanity Check

Suppose you want to prove that a program A is uncomputable. Which of the following should you do?

(a) Show that A can be solved if the halting problem could be solved.

(b) Show that the halting problem could be solved if A could be solved.

Solution:

Option b. We know that the halting problem cannot be solved, so this can help us reach a contradiction. The flow of logic will be something like: If A can be solved, then we could use it to solve the halting problem. Yet we know that we can't solve the halting problem, which creates a contradiction. Thus A must be uncomputable.

3 Code Reachability

Consider triplets (M,x,L) where

```
M is a Java program
x is some input
L is an integer
```

and the question of: if we execute $M(x)$, do we ever hit line L?

Prove this problem is undecidable.

Solution:

Suppose we had a procedure that could decide the above. Consider the following program for deciding whether $M(x)$ halts:

```
main():
run M(x);
print('hello')
```

Then we ask, is `print('hello')` ever executed?

If so, it means that $M(x)$ halts. Otherwise, $M(x)$ infinite looped.

4 Hello World!

Determine the computability of the following tasks. If it's not computable, write a reduction or self-reference proof. If it is, write the program.

(a) You want to determine whether a program P on input x prints "Hello World!". Is there a computer program that can perform this task? Justify your answer.

(b) You want to determine whether a program P prints "Hello World!" before running the kth line in the program. Is there a computer program that can perform this task? Justify your answer.

(c) You want to determine whether a program P prints "Hello World!" in the first k steps of its execution. Is there a computer program that can perform this task? Justify your answer.

Solution:

(a) Uncomputable. We will reduce `TestHalt` to `PrintsBoo`(P,x).

```
P'(x):
  run P(x) while suppressing print statements
  print("Hello World!")

TestHalt(P, x):
  if PrintsBoo(P', x):
    return true
  else:
    return false
```

If `PrintsBoo` exists, `TestHalt` must also exist by this reduction. Since `TestHalt` cannot exist, `PrintsBoo` cannot exist.

(b) Uncomputable. Reduce `PrintsBoo`(P,x) from part (a) to this program `PrintsBooByK`(P,x,k).

```
PrintsBoo(P, x):
  # Find all "return" statements in P and put these
  # line numbers in set return_statements
  return_statements = []
  for i in range(len(P)):
    if isReturnStatement(i):
      return_statements.append(i)

  # For each "return" statement, check if "Hello World!"
  # is printed
  For r in return_statements:
    if PrintsBooByK(P, x, r):
      return true
  return false
```

(c) Computable. You can simply run the program until k steps are executed. If P has halted by then, return true. Else, return false.