CS 70 Discrete Mathematics and Probability Theory
Spring 2018 Satish Rao and Babak Ayazifar **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 $\mathscr{F}$:

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

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 256, 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 $P_1, P_2, \dots$. We can label each root by a pair (i, j) corresponding to the j-th root of polynomial P_i (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 Countability Introduction

(a) Do $(0,1)$ and $\mathbb{R}_+ = (0,\infty)$ have the same cardinality? If so, give an explicit bijection (and prove that it's a bijection). If not, then prove that they have different cardinalities.

(b) Is the set of English strings countable? (Note that the strings may be arbitrarily long, but each string has finite length.) If so, then provide a method for enumerating the strings. If not, then use a diagonalization argument to show that the set is uncountable.

(c) Consider the previous part, except now the strings are drawn from a countably infinite alphabet $\mathscr{A}$. Does your answer from before change? Make sure to justify your answer.

Solution:

(a) Yes, they have the same cardinality. Consider the bijection $f : (0,1) \to (0,\infty)$ given by

$$f(x) = \frac{1}{x} - 1.$$

f is one-to-one: suppose that $f(x) = f(y)$. Then,

$$\begin{aligned} \frac{1}{x} - 1 &= \frac{1}{y} - 1, \\ \frac{1}{x} &= \frac{1}{y}, \\ x &= y. \end{aligned}$$

Hence, f is one-to-one.

f is onto: take any $y \in \mathbb{R}_+$. Let $x = 1/(1+y) \in (0,1)$. Then,

$$f(x) = \frac{1}{1/(1+y)} - 1 = 1 + y - 1 = y,$$

so f maps x to y. Hence, f is onto.

We have exhibited a bijection from $(0,1)$ to $\mathbb{R}_+$, so they have the same cardinality. (In fact, they are both uncountable.)

(b) Countable.

We can enumerate the strings as follows: list all strings of length 1, and then all strings of length 2, and then strings of length 3, and so forth. At each step, there are only finitely many strings of a particular length ℓ, so this algorithm never gets "stuck" at any length.

(c) No, the strings are still countable, although the enumeration requires a little more finesse. Notice that if we tried to list all strings of length 1, we would be stuck forever, since the alphabet is infinite! On the other hand, if we try to restrict our alphabet and only print out strings containing the first character $a \in \mathscr{A}$, we would also have a similar problem: the list

$$a, aa, aaa, \ldots$$

also does not end.

Let $\mathscr{A} = \{a_1, a_2, \ldots\}$ denote the alphabet. (We are making use of the fact that the alphabet is countably infinite when we assume there is such an enumeration.) The idea is to restrict *both* the length of the string and the characters we are allowed to use:

(a) List all strings containing only a_1 which are of length at most 1.

(b) List all strings containing only characters in $\{a_1, a_2\}$ which are of length at most 2.

(c) List all strings containing only characters in $\{a_1, a_2, a_3\}$ which are of length at most 3.

(d) Proceed onwards.

At each step, we have restricted ourselves to a finite alphabet with a finite length, so each step is guaranteed to terminate. To show that the enumeration is complete, consider any string s of length ℓ; since the length is finite, it can contain at most ℓ distinct a_i from the alphabet. Let k denote the largest index of any a_i which appears in s. Then, s will be listed in step $\max(k, \ell)$, so it appears in the enumeration.

In fact, the above idea should look somewhat familiar to you: this is the same method we used to show that $\mathbb{Z} \times \mathbb{Z}$ (and therefore $\mathbb{Q}$) is countable. We produce a table

$$\begin{array}{cccc} & \{a_1\} & \{a_1, a_2\} & \cdots \\ 1 & \swarrow & \swarrow & \swarrow \\ 2 & \swarrow & \swarrow & \\ \vdots & \swarrow & & \end{array}$$

where the column represents which characters we are allowed to use in the string, and the row represents the maximum length of the string. To enumerate all of the strings, we proceed diagonally through the table.

3 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 **??**. 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.

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 `PrintsHW`(P,x).

```
TestHalt(P, x):
  P'(x):
    run P(x) while suppressing print statements
    print("Hello World!")
  if PrintsHW(P', x):
    return true
  else:
    return false
```

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

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

```
PrintsHW(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 PrintsHWByK(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.

The reason that part (b) is uncomputable while part (c) is computable is that it's not possible to determine if we ever execute a specific line because this depends on the logic of the program, but the number of computer instructions can be counted.