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) The integers which divide 8.

(b) The integers which 8 divides.

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

(d) The set of strings over the English alphabet. (Note that the strings may be arbitrarily long, but each string has finite length. Also the strings need not be real English words.)

(e) Computer programs that halt. *Hint: How can we represent a computer program?*

(f) The set of finite-length strings drawn from a countably infinite alphabet, $\mathscr{A}$.

Solution:

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

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

(c) Uncountably infinite. We use 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 all $i \in \mathbb{N}$. We claim that the function g is not in our finite list of functions. Suppose for contradiction that it were, 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 argument, i.e. $f_n(n) \neq g(n)$, because by our construction $g(n) = f_n(n) + 1$. Contradiction!

(d) Countably infinite. The English language has a finite alphabet (52 characters if you count only lower-case and upper-case letters, or more if you count special symbols – either way, the alphabet is finite).

We will now enumerate the strings in such a way that each string appears exactly once in the list. We will use the same trick as used in Lecture note 10 to enumerate the elements of $\{0,1\}^*$ We get our bijection by setting $f(n)$ to be the n-th string in the list. List all strings of length 1 in lexicographic order, and then all strings of length 2 in lexicographic order, and then strings of length 3 in lexicographic order, and so forth. Since at each step, there are only finitely many strings of a particular length ℓ, any string of finite length appears in the list. It is also clear that each string appears exactly once in this list.

(e) 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. (Note also that this result together with the previous one in (g) implies that not every function from $\mathbb{N}$ to $\mathbb{N}$ can be written as a program.)

(f) Countably infinite. Let $\mathscr{A} = \{a_1, a_2, \dots\}$ denote the alphabet. (We are making use of the fact that the alphabet is countably infinite when we assume there is such an enumeration.) We will provide two solutions:

Alternative 1: We will enumerate all the strings similar to that in part (b), 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

$$a, aa, aaa, \dots$$

also does not end.

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 and have not been listed before.

(c) List all strings containing only characters in $\{a_1, a_2, a_3\}$ which are of length at most 3 and have not been listed before.

(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. Further, since we are listing only those strings that have not appeared before, each string appears exactly once in the listing.

Alternative 2: We will encode the strings into ternary strings. Recall that we used a similar trick in Lecture note 10 to show that the set of all polynomials with natural coefficients is countable. Suppose, for example, we have a string: $S = a_5a_2a_7a_4a_6$. Corresponding to each of the characters in this string, we can write its index as a binary string: $(101, 10, 111, 100, 110)$. Now, we can construct a ternary string where "2" is inserted as a separator between each binary string. Thus we map the string S to a ternary string: 101210211121002110. It is clear that this mapping is injective, since the original string S can be uniquely recovered from this ternary string. Thus we have an injective map to $\{0,1,2\}^*$. From Lecture note 10, we know that the set $\{0,1,2\}^*$ is countable, and hence the set of all strings with finite length over $\mathscr{A}$ is countable.

2 Countability Basics

(a) Is $f : \mathbb{N} \to \mathbb{N}$, defined by $f(n) = n^2$, an injection (one-to-one)? Briefly justify.

(b) Is $f : \mathbb{R} \to \mathbb{R}$, defined by $f(x) = x^3 + 1$, a surjection (onto)? Briefly justify.

(c) The Bernstein-Schroder theorem states that, if there exist injective functions $f : A \to B$ and $g : B \to A$ between the sets A and B, then a bijection exists between A and B. Use this to demonstrate that $(0,1)$ and $\mathbb{R}_+ = (0,\infty)$ have the same cardinality by defining appropriate injections.

Solution:

(a) Yes. One way to illustrate is by drawing the one-to-one mapping from n to n^2. More formally we can show that the preimage is unique by showing that $m \neq n \implies f(m) \neq f(n)$.

Alternative 1: Suppose $m \neq n$. Then without loss of generality let $m > n$. Thus implies that $f(m) = m^2 > n^2 = f(n)$ and hence $f(m) \neq f(n)$.

Alternative 2: We'll do a proof by contraposition. $f(m) = f(n) \implies m = n$.

$$f(m) = f(n) \implies m^2 = n^2 \implies m^2 - n^2 = 0 \implies (m-n)(m+n) = 0 \implies m = \pm n$$

Since n can't be negative, we have an injection.

(b) Yes. For any $y \in \mathbb{R}$, there exists a corresponding value $x \in \mathbb{R}$ such that $y = f(x)$. To see this take $x = \sqrt[3]{y-1}$.

(c) The injection from $(0,1)$ to $(0,\infty)$ is trivial; consider the function $f:(0,1)\to(0,\infty)$ given by

$$f(x)=x.$$

It is easy to see that f is injective.

For the other way, consider the function $g:(0,\infty)\to(0,1)$ given by

$$g(x)=\frac{1}{x+1}.$$

To see that g is injective, suppose $g(x)=g(y)$. Then

$$\frac{1}{x+1}=\frac{1}{y+1} \implies x=y.$$

Hence g is injective. Thus we have an injective function from $(0,1)$ to $(0,\infty)$ and an injective function from $(0,\infty)$ to $(0,1)$. By the Bernstein-Schroder theorem there exists a bijection from $(0,1)$ to $(0,\infty)$ and hence they have the same cardianlity.

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:

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.

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):
  for i in range(len(P)):
    if PrintsHWByK(P, x, i):
      return true
  return false
```

(c) Computable. You can simply run the program until k steps are executed. If P has printed "Hello World!" 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.