CS 70 Discrete Mathematics and Probability Theory
Fall 2026 Hug, Sabin

HW 07

1 Countability Basics

Note 12

(a) Prove that $f : \mathbb{N} \to \mathbb{N}$, defined by $f(n) = n^2$, is an injection (one-to-one). Recall that f is injective if and only if for all $m, n \in \mathbb{N}$, $m \neq n$ implies $f(m) \neq f(n)$.

(b) Prove that $f : \mathbb{R} \to \mathbb{R}$, defined by $f(x) = x^3 + 1$, is a surjection (onto). Recall that f is surjective if and only if for every $y \in \mathbb{R}$, there exists an $x \in \mathbb{R}$ such that $f(x) = y$.

(c) In class, we showed that $(0,1)$ is uncountable. In this subpart, we will prove that $\mathbb{R}_+ = (0,\infty)$ is uncountable.

(i) Define an injection $f : (0,1) \to \mathbb{R}_+$.

(ii) Define an injection $g : \mathbb{R}_+ \to (0,1)$.

(iii) Explain why the Cantor–Bernstein theorem implies that $\mathbb{R}_+$ is uncountable.

Solution:

(a) We show that $m \neq n \implies f(m) \neq f(n)$ for all $m, n \in \mathbb{N}$. Since the domain of f is $\mathbb{N}$, both m and n are nonnegative.

Alternative 1: Suppose $m \neq n$. Then without loss of generality let $m > n$. Since $m > n \geq 0$, this 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 $m, n \in \mathbb{N}$ are nonnegative, $m = -n$ only when $m = n = 0$, so in either case $m = n$. Hence f is an injection.

(b) For any $y \in \mathbb{R}$, we need some $x \in \mathbb{R}$ such that $f(x) = y$. Take $x = \sqrt[3]{y-1}$, which is a real number. Then

$$f(x) = \left(\sqrt[3]{y-1}\right)^3 + 1 = (y-1) + 1 = y.$$

(c) (i) 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.

(ii) 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.

(iii) We have an injective function from $(0,1)$ to $(0,\infty)$ and an injective function from $(0,\infty)$ to $(0,1)$. The Cantor-Bernstein 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. By the theorem there exists a bijection from $(0,1)$ to $(0,\infty)$ and hence they have the same cardinality. Since $(0,1)$ is uncountable, $\mathbb{R}_+ = (0,\infty)$ must also be uncountable.

2 Computability Basics

Note 13

Consider the problem of determining if a function F returns `TRUE`. That is, given a function F that takes inputs from some (possibly infinite) set $\mathscr{X}$, we want to know if there is any input $x \in \mathscr{X}$ such that $F(x)$ returns `TRUE`. We will prove that this problem is uncomputable.

(a) We will use proof by contradiction. Suppose there exists a program `ReturnsTrue` that determines whether a function returns `TRUE` on some input $x \in \mathscr{X}$. What are the inputs and outputs of `ReturnsTrue`?

(b) We will reach a contradiction by solving the Halting Problem. What hypothetical program would solve the Halting Problem? What are its inputs and outputs?

(c) Fix a program P and an input x, and consider the following program:

```
def F(y):
    P(x)
    return TRUE
```

Will $F(y)$ return `TRUE` if $P(x)$ halts? Will it return `TRUE` if $P(x)$ loops forever?

(d) Use `ReturnsTrue` to solve the Halting Problem, and conclude that `ReturnsTrue` cannot exist.

Solution:

(a) `ReturnsTrue` takes as input a function F whose inputs come from $\mathscr{X}$. It returns `TRUE` if there is some $x \in \mathscr{X}$ such that $F(x)$ returns `TRUE`, and `FALSE` if no such x exists.

(b) The program `TestHalt` would solve the Halting Problem. It takes as input a program P and an input x, and returns `TRUE` if $P(x)$ halts and `FALSE` if $P(x)$ runs forever.

(c) If $P(x)$ halts, then $F(y)$ finishes running $P(x)$ and returns `TRUE`, for every input y. If $P(x)$ loops forever, then $F(y)$ never gets past its first line, so it never returns `TRUE` on any input y.

(d) We can use `ReturnsTrue` to write `TestHalt`:

```
def TestHalt(P, x):
    def F(y):
        P(x)
        return TRUE
    return ReturnsTrue(F)
```

By part (c), if $P(x)$ halts, then $F(y)$ returns `TRUE` on every input y, so `ReturnsTrue`(F) returns `TRUE`. On the other hand, if $P(x)$ does not halt, then F never returns anything on any input, so `ReturnsTrue`(F) returns `FALSE`. Thus `TestHalt`(P,x) returns `TRUE` exactly when $P(x)$ halts, so it solves the Halting Problem. But the Halting Problem is uncomputable, which is a contradiction. Therefore `ReturnsTrue` cannot exist.

3 Countability and the Halting Problem

Note 12
Note 13

Using methods from countability, we will prove the Halting Problem is uncomputable.

(a) What is a reasonable representation for a computer program? Using this definition, show that the set of all programs is countable. *(Hint: Machine Code)*

(b) The Halting Problem only considers programs which take a finite length input. Show that the set of all finite-length inputs is countable.

(c) Assume that you have a program that tells you whether or not a given program halts on a specific input. Since the set of all programs and the set of all inputs are countable, we can enumerate them and construct the following table.

	x_1	x_2	x_3	x_4	$\cdots$
p_1	H	L	H	L	$\cdots$
p_2	L	L	L	H	$\cdots$
p_3	H	L	H	L	$\cdots$
p_4	L	H	L	L	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

An H (resp. L) in the ith row and jth column means that program p_i halts (resp. loops) on input x_j. Now write a program that is not within the set of programs in the table above.

(d) Find a contradiction between parts (a) and (c) to show that the Halting Problem can't be solved.

Solution:

(a) We can represent a computer program by its machine code, which is a finite-length binary string. This gives an injection from the set of all programs into $\{0,1\}^*$, the set of finite-length binary strings. Since $\{0,1\}^*$ is countable (Note 12), the set of all programs is countable.

(b) Every finite-length input can also be represented by a finite-length binary string, so the set of all finite-length inputs also injects into $\{0,1\}^*$. Therefore the set of all finite-length inputs is countable.

(c) Call the assumed program `TestHalt`: `TestHalt`(P,x) returns `TRUE` if $P(x)$ halts and `FALSE` if $P(x)$ loops. We use it to flip the diagonal of the table:

```
def Q(x):
    j = index of x in the list x_1, x_2, ...
    if TestHalt(p_j, x_j):
        loop forever
    else:
        halt
```

Q can find j and p_j by listing the inputs and programs in the same order used for the table. For every j, Q does the opposite of p_j on input x_j: if $p_j(x_j)$ halts, then $Q(x_j)$ loops, and if $p_j(x_j)$ loops, then $Q(x_j)$ halts. So Q differs from every program in the table.

(d) By part (a), the table lists every program, so $Q = p_n$ for some n. Consider the diagonal entry in row n and column n. If it is H, then $p_n(x_n)$ halts, so by construction $Q(x_n)$ loops; but $Q = p_n$, so $p_n(x_n)$ loops. If it is L, then $p_n(x_n)$ loops, so $Q(x_n)$ halts, meaning $p_n(x_n)$ halts. Either way we have a contradiction, so `TestHalt` cannot exist, and the Halting Problem cannot be solved.

4 Count It!

Note 12

For each of the following sets, 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) The set of finite-length strings drawn from a countably infinite alphabet, $\mathscr{A}$.

(f) The set of infinite-length strings over the English alphabet.

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 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 Note 12 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. 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 (d), 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:

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

2. List all strings containing only characters in $\{a_1, a_2\}$ which are of length at most 2 and have not been listed before.
3. 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.
4. 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 Note 12 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 Note 12, 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.

(f) Uncountably infinite. We can use a diagonalization argument. First, for a string s, define $s[i]$ as the i-th character in the string (where the first character is position 0), where $i \in \mathbb{N}$ because the strings are infinite. Now suppose for contradiction that we have an enumeration of strings s_i for all $i \in \mathbb{N}$: then define the string s' as $s'[i] = (\text{the next character in the alphabet after } s_i[i])$, where the character after z loops around back to a. Then s' differs at position i from s_i for all $i \in \mathbb{N}$, so it is not accounted for in the enumeration, which is a contradiction. Thus, the set is uncountable.

Alternative 1: The set of all infinite strings containing only as and bs is a subset of the set we're counting. We can show a surjection from this subset onto the real interval $[0, 1]$, which proves the uncountability of the subset and therefore entire set as well: given an infinite string of as and bs, replace the as with 0s and bs with 1s and prepend '0.' to the string, which produces the binary expansion of a number in $[0, 1]$. Every number in $[0, 1]$ has a binary expansion, so every number is hit. (This is not a bijection, since some numbers have two expansions, such as $0.0111\ldots = 0.1000\ldots$, but a surjection is enough: if the subset were countable, listing the images of its elements would make $[0, 1]$ countable.)

5 Unprogrammable Programs

Note 13

Prove whether the programs described below can exist or not.

(a) A program $P(F,x,y)$ that returns true if the program F outputs y when given x as input (i.e. $F(x) = y$) and false otherwise.

(b) A program P that takes two programs F and G as arguments, and returns true if for all inputs x, F halts on x iff G halts on x (and returns false if this equivalence is not always true).

Hint: Use P to solve the halting problem, and consider defining two subroutines to pass in to P, where one of the subroutines always halts.

Solution:

(a) P cannot exist, for otherwise we could solve the halting problem:

```
def Halt(F, x):
    def Q(x):
        F(x)
        return 0
    return P(Q, x, 0)
```

`Halt` defines a subroutine Q that first simulates F and then returns 0, that is $Q(x)$ returns 0 if $F(x)$ halts, and nothing otherwise. Knowing the output of $P(Q,x,0)$ thus tells us whether $F(x)$ halts or not.

(b) We solve the halting problem once more:

```
def Halt(F, x):
    def Q(y):
        halt
    def R(y):
        if y == x:
            F(x)
        else:
            halt
    return P(Q, R)
```

Q is a subroutine that always halts. R is a subroutine that halts on every input except x, and runs $F(x)$ on input x when handed x as an argument.

Knowing if Q and R halt on the same inputs boils down to knowing whether F loops on x (since that is the only case in which they could possibly differ). Thus, if $P(Q,R)$ returns "True", then we know they behave the same on all inputs and F must halt on x, and if $P(Q,R)$ returns "False", then we know they differ on some input (which must be x) and F must loop on x. Thus, if P exists, this algorithm solves the halting problem, which is a contradiction. Therefore, P cannot exist.