

1 Countability Intro

Note 12

A function $f : A \rightarrow B$ maps elements from set A to set B .

f is *injective* if it maps distinct elements to distinct elements, and *surjective* if every element in B is mapped to by some element in A . If f is both injective and surjective, it is *bijective*, and the sets A and B are said to have the same *cardinality* (size). The cardinality of a set is denoted by $|A|$.

f is bijective if and only if there exists an inverse function $f^{-1} : B \rightarrow A$ such that $f^{-1}(f(a)) = a$ for all $a \in A$ and $f(f^{-1}(b)) = b$ for all $b \in B$.

Countability: Formal notion of different kinds of infinities.

- *Countable*: able to enumerate in a list (possibly finite, possibly infinite)
- *Countably infinite*: able to enumerate in an infinite list; that is, there is a bijection with $\mathbb{N}$.

To show that there is a bijection, the *Cantor–Bernstein theorem* says that it is sufficient to find two injections, $f : S \rightarrow \mathbb{N}$ and $g : \mathbb{N} \rightarrow S$. Intuitively, this is because an injection $f : S \rightarrow \mathbb{N}$ means $|S| \leq |\mathbb{N}|$, and an injection $g : \mathbb{N} \rightarrow S$ means $|\mathbb{N}| \leq |S|$; together, we have $|\mathbb{N}| = |S|$.

- *Uncountably infinite*: a set with a cardinality larger than $|\mathbb{N}|$. Therefore, there is no bijection with $\mathbb{N}$ and the set can't be listed out.

Use *Cantor diagonalization* to prove uncountability through contradiction; the classic example is the set of reals in $[0, 1]$:

$\mathbb{N}$	$[0, 1]$
0	0 . 7 3 2 0 5 0 ...
1	0 . 4 1 4 2 1 3 ...
2	0 . 6 1 8 0 3 3 ...
3	0 . 1 8 2 8 1 8 ...
4	0 . 1 4 1 5 9 2 ...
5	0 . 5 7 7 2 1 5 ...
$\vdots$	$\vdots$
?	0 . 8 2 9 9 1 6 ...

To have a bijection with $\mathbb{N}$, we must be able to assign a natural number to every real number, as above. In other words, every real number must eventually appear on this (infinite) list. However, if we change the digits along the diagonal and compile that into a new real number, it is different from every single element in the list in at least one place, so it's not in the list—this is a **contradiction**, and it implies that there is at least one real number “left over”. Hence, the cardinality of $\mathbb{R}$ is strictly greater than $\mathbb{N}$!

Sometimes it can be easier to prove countability (or uncountability) through bijections with other countable (or uncountable) sets, respectively. Common countable sets include $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{N} \times \mathbb{N}$, finite length bitstrings, etc. Common uncountable sets include $[0, 1]$, $\mathbb{R}$, infinite length bitstrings, etc.

(a) Read through the previous page's description of Cantor diagonalization. In your own words, why do we know that the number created by changing the digits along the diagonal is not in the list?

(b) True or false: the new number created is a real number in $[0, 1]$ that is not on the list.

(c) Your friend is confused about how Cantor diagonalization doesn't apply to the set of natural numbers. They argue that natural numbers can be thought of as an infinite length string of digits, by padding each number with an infinite number of zeroes to the left. If we then assume by contradiction that we can list out the set of natural numbers with the padded 0's, we can change the digits along a diagonal, to create a new natural number not in the list.

0	...	0	0	0	0	①
1	...	0	1	2	③	4
2	...	5	2	⑧	2	3
3	...	9	④	3	2	1
⋮				⋮		
?	...	1	5	9	4	2

What is wrong with this argument?

2 Counting Cartesian Products

Note 12

For two sets A and B , define the cartesian product as $A \times B = \{(a, b) : a \in A, b \in B\}$.

(a) Given two countable sets A and B , prove that $A \times B$ is countable.

(b) Given a finite number of countable sets $A_1, A_2, \dots, A_n$, prove that

$$A_1 \times A_2 \times \dots \times A_n$$

is countable.

(c) Consider a countably infinite number of finite sets: $B_1, B_2, \dots$ for which each set has at least 2 elements. Prove that $B_1 \times B_2 \times \dots$ is uncountable.

3 Counting Functions

Note 12

Are the following sets countable or uncountable? Prove your claims.

- (a) The set of all functions f from $\mathbb{N}$ to $\mathbb{N}$ such that f is non-decreasing. That is, $f(x) \leq f(y)$ whenever $x \leq y$.

- (b) The set of all functions f from $\mathbb{N}$ to $\mathbb{N}$ such that f is non-increasing. That is, $f(x) \geq f(y)$ whenever $x \leq y$.

4 Computability Intro

Note 13

Computability: The main focus is on the Halting problem, and programs that provably cannot exist.

The *Halting problem* is the problem of determining whether **any arbitrary** program P run on input x ever halts, or whether it loops forever. It turns out that there does not exist any program that solves this problem.

Using this information, we can prove that other problems also cannot be solved by a computer program, through the use of *reductions*. The main idea is to show that if a given problem can be solved by a computer program TestX , then the Halting problem can also be solved by a computer program TestHalt that uses TestX as a subroutine.

Here is an example of a template that we will often use for reductions. Suppose we want to show that a program TestX does not exist, where $\text{TestX}(P', y)$ tries to determine whether a program P' on input y does some task $\mathcal{X}$ (i.e. it outputs “True” if $P'(y)$ does the task $\mathcal{X}$, and it outputs “False” if $P'(y)$ does not do the task $\mathcal{X}$). We can define TestHalt as follows (in pseudocode):

```
def TestHalt(P, x):  
    def P'(y):  
        run P(x)  
        do  $\mathcal{X}$   
    return TestX(P', y) # for some given y
```

Then we will show that if TestX exists, then TestHalt would correctly solve the halting problem.

Note that reductions in CS70 will generally look like this template. but more complex reductions will require more sophisticated programs—you’ll learn more about this in classes like CS170 and CS172.

(a) Consider the reduction template given above. Let’s break down what it’s doing.

We follow an argument by contradiction—we assume that there is a program $\text{TestX}(P', y)$ that is able to determine whether another program P' on input y does some task $\mathcal{X}$.

There are two cases: either $P(x)$ halts, or it loops forever. We’d like to show that TestHalt as defined above returns the correct answer in both of these cases.

(i) Suppose $P(x)$ halts. What does TestHalt return, and why?

(ii) Suppose $P(x)$ loops forever. What does TestHalt return, and why?

(iii) What does this tell us about the existence of TestX ? Briefly justify your answer.

5 Hello World!

Note 13

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. Throughout this problem, you are allowed to execute programs while suppressing their print statements.

- (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!" while or before running the k th 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.

6 Code Reachability

Note 13

Consider triplets (M, x, L) where

- M is a 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 that this problem is uncomputable.