CS 70 Discrete Mathematics and Probability Theory
Fall 2025 Sabin, Hug

DIS 7B

Countability Intro

Note 11

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, $f : S \to \mathbb{N}$ and $g : \mathbb{N} \to S$. Intuitively, this is because an injection $f : S \to \mathbb{N}$ means $|S| \leq |\mathbb{N}|$, and an injection $g : \mathbb{N} \to S$ means $|\mathbb{N}| \leq |S|$; together, we have $|\mathbb{N}| = |S|$.
- *Uncountably infinite*: unable to be listed out

  Use *Cantor diagonalization* to prove uncountability through contradiction.

Sometimes it can be easier to prove countability/uncountability through bijections with other countable/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.

1 Counting Cartesian Products

Note 11

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 $A_1, A_2, \ldots, A_n$, prove that

$$A_1 \times A_2 \times \cdots \times A_n$$

is countable.

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

2 Computability Intro

Note 12

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

The *Halting problem* is the problem of determining whether a 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.

The primary template we'll use for this course is as follows. Suppose we want to show that a program `TestX` does not exist, where `TestX(Q, y)` tries to determine whether a program Q on input y does some task $\mathscr{X}$ (i.e. it outputs "True" if $Q(y)$ does the task $\mathscr{X}$, and it outputs "False" if $Q(y)$ does not do the task $\mathscr{X}$). We can define `TestHalt` as follows (in pseudocode):

```
def TestHalt(P, x):
    def Q(y):
        run P(x)
        do 𝒳
    return TestX(Q, y)  # for some given y
```

Note that this template will be sufficient for our purposes in CS70, 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 `TestX(Q, y)` that is able to determine whether another program Q on input y does some task $\mathscr{X}$.

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.

3 Hello World!

Note 12

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 surpressing 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 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.

4 Code Reachability

Note 12

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.