COMPSCI 230: Discrete Mathematics for Computer Science April 1, 2019

Lecture 20

Lecturer: Debmalya Panigrahi *Scribe: Erin Taylor*

1 Overview

In this lecture, we prove that certain infinite sets are uncountable using a technique known as diagonalization. We also study the halting problem.

2 Infinite Sets

2.1 Countability

Last lecture, we introduced the notion of countably and uncountably infinite sets. Intuitively, countable sets are those whose elements can be listed in order. In other words, we can create an infinite sequence containing all elements of a countable set. Formally, countable sets are those that have a bijection with the set of all natural numbers $\mathbb{N}$ (a.k.a. positive integers $\mathbb{Z}^+$). In some sense, this is the "smallest" an infinite set can be. We formalize this intuition with the following Lemma.

Lemma 1. *For any infinite set A and any countable set B, A surj B.*

Proof. We will first prove the following statement: For any infinite set A, A surj $\mathbb{N}$. Because A is infinite, there exists some element of A which we denote by a_1. Additionally, there exists an element $a_2 \in A \setminus \{a_1\}$. We can continue this process. For any $n \in \mathbb{N}$ pick $a_n \in A \setminus \{a_1, \dots a_{n-1}\}$. Let $S = (a_1, a_2, \dots)$ be the infinite sequence of elements of A generated by the above procedure. We can use this to create a surjection $f : A \to \mathbb{N}$:

- $f(a_i) = i \quad \forall i \in \mathbb{N}$.
- $f(a) = 1 \quad \forall a \in A,\ a \neq a_i\ \forall i \in \mathbb{N}$.

f is a surjection since every natural number except 1 is mapped exactly once by the corresponding a_i, and 1 is mapped at least once by a_1. Since B is countable, $\mathbb{N}$ bij B implies $\mathbb{N}$ surj B. If A surj $\mathbb{N}$ and $\mathbb{N}$ surj B, then A surj B (see Lecture 19). □

2.2 Power Sets of Infinite Sets

In the last lecture, we showed that $2^{\mathbb{N}}$ was not countable by proving that there does not exist a surjection between a set and its power set. In some sense, this implies the collection of all subsets of a set is strictly larger than the set itself. We will formalize this notion.

Definition 1. *Let A and B be two sets. We say "A* strict *B" if and only if $\neg(A$ surj $B)$.*

Notice that if A and B are finite sets, then A strict B means A is smaller than B, i.e., $|A| < |B|$.

Using this fact, we could create an infinite sequence of larger and larger infinite sets by taking power sets.

$$\mathbb{N} \text{ strict } 2^{\mathbb{N}} \text{ strict } 2^{2^{\mathbb{N}}} \text{ strict } 2^{2^{2^{\mathbb{N}}}} \text{ strict } \ldots$$

We are mostly interested in the first two 'tiers' of this hierarchy. We have already seen many examples of countable sets that are useful in computer science: the natural numbers, the integers, the rational numbers, etc. Next, we will see more examples of uncountable sets that arise in both mathematics and computer science frequently.

2.3 Diagonalization

We know that $2^{\mathbb{N}}$ is an uncountable set. Consider a subset of the natural numbers: $S \subseteq \mathbb{N}$. We can represent any such S as an infinite length binary string, where the bit at the i^{th} index is 1 if $i \in S$, and 0 otherwise.

For example, suppose $S = \{1,\ 3,\ 5,\ 6\}$. Then, we could represent S in binary as $10101100\ldots$. We denote the set of all finite binary strings by $\{0,1\}^*$, and the set of all infinite binary strings by $\{0,1\}^\omega$. (Note that $\{0,1\}^*$ is an infinite set, but each element of this set is a binary string of finite length. This is different from $\{0,1\}^\omega$, which is not only an infinite set, but each element of the set is also a binary string of infinite length.)

The above representation is a bijection with $2^{\mathbb{N}}$. This implies that $\{0,1\}^\omega$ is uncountable. We frequently work with binary strings in computer science. In reality, computers have finite memory so we could only ever deal with finite strings, but almost all formal models of computation deal with infinite strings. We can show $\{0,1\}^\omega$ is uncountable with a direct argument, rather than creating a bijection with $2^{\mathbb{N}}$.

First, note that $\{0,1\}^*$ is countable. Observe that we can think of a finite binary string as a natural number expressed in binary, and this immediately yields a bijection.

Theorem 2. *The set of infinite binary strings, $\{0,1\}^\omega$, is uncountable.*

Proof. Suppose $\{0,1\}^\omega$ was countable. Then there is a bijection: $f : \mathbb{N} \to \{0,1\}^\omega$. Let $s_i = f(i)\ \forall i \in \mathbb{N}$. For example, we could list s out as follows.

$$\begin{array}{ll}
s_1 : 01101101\ldots & s_1 : 0\mathbf{1}101101\ldots \\
s_2 : 11000101\ldots & s_2 : 1\mathbf{1}000101\ldots \\
s_3 : 11110001\ldots & s_3 : 11\mathbf{1}10001\ldots \\
s_3 : 00100101\ldots & s_3 : 001\mathbf{0}0101\ldots \\
s_4 : 01101011\ldots & s_4 : 0110\mathbf{1}011\ldots \\
\vdots & \vdots
\end{array}$$

We can construct a new infinite-length binary string by flipping each bit down the diagonal (flip the bits shown in red), giving us a new infinite-length binary string s^*. In the above example, $s^* = 100110\ldots$.

By construction, s^* differs from each s_n, since their n^{th} digits differ (red in the example). Thus, s^* is not in the bijection , a contradiction. Therefore, the number of infinite-length binary strings is uncountable. □

Recall that each real number has an infinite length decimal expansion. For example:

$$1/2 = .500000\ldots$$
$$1/3 = .333333\ldots$$
$$\sqrt{2} = 1.4142\ldots$$
$$\pi = 3.14159\ldots$$

The above decimal expansion is in base 10, but we could express the decimal expansion in binary (this is just a change of base). For example, $1/2 = .1000\ldots$ and $3/4 = .1100\ldots$.

Corollary 3. $[0,1] = \{r \in \mathbb{R} : 0 \leq r \leq 1\}$ *is uncountable.*

Proof. This theorem follows from the previous discussion. Starting at the decimal point, real numbers have a infinite length binary expansion. For any real number $r \in [0,1)$, map it to the corresponding binary string representation. This function is a surjection onto $\{0,1\}^\omega$. Since $\{0,1\}^\omega$ is uncountable, this implies that $[0,1]$ is uncountable. □

Corollary 4. $\mathbb{R}$ *is uncountable.*

Proof. Define the function $f : \mathbb{R} \to [0,1]$ by

- $f(x) = x$ for all $x \in [0,1]$.
- $f(x) = 0$ for all $x \notin [0,1]$.

f is a surjection. Thus, $[0,1]$ is uncountable and $\mathbb{R}$ surj $[0,1)$ implies that $\mathbb{R}$ is uncountable. □

This gives us an easy way to show that a set in uncountable, which we summarize in the next lemma.

Lemma 5. *Any infinite set A is uncountable if A surj $\mathbb{R}$.*

3 The Halting Problem

Perhaps it is not immediately clear how the discussions of infinite sets and "sizes" of infinity are relevant to computer science. In fact, many of the same arguments are also used to show that various computational problems are not solvable, even with unlimited time, space, and resources. We will see one such example.

Consider a program. When you run a program on your computer, you would like it to terminate, whether that is by producing some desired output, prompting for more input, or throwing an error. If the program stops computation on an input, the program is said to *halt*. Before you begin to run a program on some input, you might want to know whether the program will halt or not. If it will not halt, you may choose not to run it. The *Halting Problem* is the the problem of determining whether a program will halt on a given input. We may even want to write a program that takes in another program and an input and determine whether the program would halt on the input. It turns out that this problem is undecidable: there cannot be any program that gives the correct answer to whether another program halts on a given input, for every program and every input. This was first shown by Alan Turing in 1936. We will prove his result below.

First, observe that we can think of any program as a string. Computers will compile code in a high-level language into binary in order to run it. Going forward, we will think of a program itself, and its possible input, as finite binary strings. For a given program we define its *language*: the set of all input binary strings that the program halts on.

$$lang(P_s) = \{s' : P_s \text{ halts on } s'\}.$$

Since s is a machine-readable version of the program, s itself is a binary string. We could give s as input to the program P_s. This allows us to define a set of input strings on which the program representing that input string itself does not halt.

$$\mathcal{L} = \{s : P_s \text{ does not halt on } s\}.$$

Theorem 6 (Turing, 1936)**.** *There is no program P_{s_0} such that $\mathcal{L} = lang(P_{s_0})$.*

This theorem says that there is no program that halts on exactly the strings in $\mathcal{L}$. In other words, a computer program cannot decide, for every other computer program, whether it halts when given its own string representation as input.

Proof. We will prove this theorem by contradiction. Suppose $\mathcal{L} = lang(P_{s_0})$. Then,

$$\forall s.\, s \in \mathcal{L} \;\; \textit{iff} \;\; s \in lang(P_{s_0}).$$

We also know that s_0 is a string itself, so it must be that

$$s_0 \in \mathcal{L} \;\; \textit{iff} \;\; s_0 \in lang(P_{s_0}).$$

We claim this is a contradiction. $s_0 \in \mathcal{L}$ means that P_{s_0} does not halt on s_0 and $s_0 \in lang(P_{s_0})$ if and only if P_{s_0} halts on s_0. □

This theorem implies that you cant possibly write a program which always detects bugs (such as infinite loops) in other programs.

4 Summary

In this lecture, we gave examples of countable and uncountable sets. We proved sets were uncountable using a diagonalization argument. We also considered the Halting problem, and showed that diagonalization arguments are useful in showing that not all computational problems are solvable.