A Crash Course in Undecidability

Contents

1 Introduction

This document is a quick introduction to undecidability. The area of undecidability is vast and can at times be very intricate, which makes putting it into a nutshell and hitting the "main" topics nearly impossible. Nonetheless, this document attempts to summarize a few of the more important results of the theory.

2 Introduction, Preliminary Results and Notation

2.1 The Number of Turing Machines is Countably Infinite

In Section 7 of this document we discuss the different sizes of infinity. For the purposes of undecidability we need only realize the difference between a countable set and an uncountable set.

Definition 1: *A set is said to be* countable *(or* enumerable *or* denumerable*) if there is a bijective map from it to a subset of* $\mathbb{N}$*. A set that is not countable is said to be* uncountable*.*

In other words, a set is countably infinite if we can find a one-to-one and onto map between the elements of the set and the counting numbers. One thing to notice about this definition is that it also includes finite sets. Note that not all texts agree on the inclusion of finite sets in the definition of countable. In many books the term countable applies to infinite sets only. Since we will be working exclusively with infinite sets in this discussion it really does not make any difference. Also from this definition it is obvious that any subset of a countable set is also countable.

Example 1: Let $S = \{a_1, a_2, \ldots a_n\}$ be any finite set. Then S^* is countably infinite.

To see this we need only devise a way to enumerate S^*. One way is to order the set S^* by word size then in each size we order by the given order in the set S. That is, we list all of the elements of S^* as,

$$\lambda, a_1, a_2, \ldots a_n, a_1a_1, a_1a_2, a_1a_3, \ldots a_1a_n, a_2a_1, a_2a_2, a_2a_3, \ldots, a_2a_n, \ldots, a_na_n, a_1a_1a_1, \ldots$$

From this result, it is easy to establish that the set of Turing Machines is countably infinite.

Theorem 1: *The number of distinct Turing Machines is countably infinite.*

Proof. There are clearly an infinite number of Turing Machines since there is no limit to the number of transitions that are used in its definition. Another thing to note is that any Turing Machine is uniquely defined by its set of transitions. We can write any transition of a Turing Machine in binary form by using the following method. Let the set of states of the Turing Machine be

$$Q = \{q_1, q_2, q_3, \ldots, q_n\}$$

and let the set of tape symbols of the Turing Machine be

$$\Gamma = \{a_1, a_2, a_3, \ldots, a_m\}$$

If we have the transition

$$\delta(q_1, a_2) = (q_3, a_4, L)$$

we could write it using a unary representation for the subscripts and delimit them using 0's. We could also use the convention that L would be 1 and R would be 2. For example, the transition above would be written as,

$$01011011101111010$$

and the transition

$$\delta(q_5, a_1) = (q_2, a_2, R)$$

would be written as,

$$011111010110110110$$

Then to represent the entire Turing Machine we would simply concatenate all of the coded transitions. This establishes that the set of all Turing Machines is a subset (in fact, obviously a proper subset) of the set $\{0, 1\}^*$ which we know to be countable. Thus the number of Turing Machines is countable. □

And since our definition of an algorithm is that there exists a Turing Machine that can preform the algorithm the above theorem tells us that there are a countably infinite number of algorithms. Although you might think that this will be enough to keep you busy for a while, we will see that it does impose some severe restrictions on what can be done by a mechanical computing device.

Another couple points about this. You may have noticed that in the above proof we did not concern ourselves with the order of the transitions when we created the coding of the Turing Machine into 0's and 1's. So we could have the same Turing Machine counted twice, or three times or more by rearranging the transitions in the coding process. This is simply a technical point that does not affect the countability issue. We are simply saying that two Turing Machines can have the same set of transitions but have different coding. As computer scientists you should feel at home with this since as you know all too well that the same program can be coded in many different ways.

This is not the only way to code a Turing Machine into a subset of $\{0, 1\}^*$. Gödel did a similar, but different, coding process to enumerate the set of Turing Machines. This process has become known as **Gödel Numbering**. Whatever process we use, we can associate any Turing Machine M with a natural number.

Notation: Given a Turing Machine M, its associated natural number, or Gödel number, is denoted as $n(M)$.

Example 2: The Turing machine that will move right until it hits a blank, $R_\Box$ can be written as,

$$\begin{aligned}
\delta(q_0, a) &= (q_0, a, R) \\
\delta(q_0, b) &= (q_0, b, R) \\
\delta(q_0, \Box) &= (q_1, \Box, R) \\
\delta(q_1, a) &= (q_f, a, L) \\
\delta(q_1, b) &= (q_f, b, L) \\
\delta(q_1, \Box) &= (q_f, \Box, L)
\end{aligned}$$

To begin coding it we make the following alterations, we will increase the subscripts of the states all by 1, hence making q_1 the initial state and we will denote q_f as q_3. We will also make the following associations, $a_1 = \Box$, $a_2 = a$, and $a_3 = b$. With this, our transitions become,

$$\begin{aligned}
\delta(q_1, a_2) &= (q_1, a_2, R) \\
\delta(q_1, a_3) &= (q_1, a_3, R) \\
\delta(q_1, a_1) &= (q_2, a_1, R) \\
\delta(q_2, a_2) &= (q_3, a_2, L) \\
\delta(q_2, a_3) &= (q_3, a_3, L) \\
\delta(q_2, a_1) &= (q_3, a_1, L)
\end{aligned}$$

Using our coding scheme each transition would be encoded as,

$$01011010110110 \quad 0101110101110110 \quad 0101011010110$$

$$0110110111011010 \quad 011011101110111010 \quad 01101011101010$$

Giving the binary number

$$0101101011011001011101011101100101011010110011011011101101001101110111011101001101011101010$$

which represents the decimal number 878,642,009,785,702,569,885,997,802. So

$$n(R_\Box) = 878,642,009,785,702,569,885,997,802$$

2.2 Another Type of Equivalent Turing Machine

In the Linz textbook[2], the definition of a Turing Machine had a set of favorable states that was a subset of the set of states. This definition was consistent with the other automata we have studied this semester. With this definition, a Turing Machine would accept a word if the machine ended its computation, that is halted, on a favorable state. It would reject the word if the machine halted on an unfavorable state. An equivalent definition of a Turing Machine would be to remove the set of favorable states and consider the machine halting to

be favorable and the machine not halting to be unfavorable. So a word would be accepted if the Turing Machine halted on that word and not accepted if the Turing Machine did not halt on that word. Formally,

Definition 2: *A Turing Machine M accepts a word $w \in \Sigma^*$ if $M(w)$ halts and does not accept the word w if $M(w)$ does not halt.*

The equivalence of this type of Turing Machine with our definition is easily seen. If our standard machine reaches a favorable conclusion it would halt anyway. If the machine would halt in a non-favorable state we would simply create a trap-like state that put the machine into an infinite loop.

When using this type of Turing Machine we get a similar, but slightly different, definition of the computation of a function.

Definition 3: *A Turing Machine M* computes *a function $f : \Sigma^* \to \Sigma^*$ if for each $x \in Dom(f)$, $M(x) = f(x)$ and for each $x \notin Dom(f)$, $M(x)$ does not halt. Here, $Dom(f)$ represents the domain of the function f, which is of course a subset of Σ^*.*

More specifically, we make a distinction between computable functions depending on if their domain is all of Σ^* or only part of Σ^*.

Definition 4: *A function $f : \Sigma^* \to \Sigma^*$ is said to be* Partial Turing Computable *if $Dom(f) \subseteq \Sigma^*$ and there is a Turing Machine that computes it.*

Definition 5: *A function $f : \Sigma^* \to \Sigma^*$ is said to be* Turing Computable *if it is Partial Turing Computable and if $Dom(f) = \Sigma^*$. That is, if $Dom(f) = \Sigma^*$ and there a Turing Machine that computes f.*

We can now define what it means for a language to be decidable.

Definition 6: *Let $L \subseteq \Sigma^*$ be a language, the characteristic function of L is defined to be,*

$$\eta_L(w) = \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$

Definition 7: *A Language L is called* Decidable *if there is a Turing Machine that computes the characteristic function of L.*

Note that the characteristic function of any language L has a domain of Σ^*. So if a language is Decidable then its characteristic function is Turing Computable, that is, there is a Turing Machine M that halts on every input word from Σ^* and outputs either a 1 or 0 depending on $w \in L$ or $w \notin L$ respectively. For this reason, decidable languages are sometimes called *Turing Computable* languages. We will use the term *Decidable*.

Definition 8: *A language L is called* Semidecidable *if there is a Turing Machine M that outputs 1 if $w \in L$ and does not halt on any $w \notin L$.*

So the big difference between Decidable and Semidecidable is that for a decidable language the associated Turning Machine must halt on every input and output either a 0 or 1 and for a semidecidable language the associated Turning Machine must halt and output a 1 only if the word was in the language. In other words, for a decidable language the associated

Turning Machine must be able to tell if a word is in the language and if a word is not in the language whereas for a semidecidable language the associated Turning Machine need only be able to tell if a word is in the language.

2.3 Some Closure Properties

Although some of these properties rely on the negative result of the Halting Problem we include them all here for convenience. We will simply state these without proof.

Theorem 2: *If a language L is decidable then its complement $\overline{L}$ is decidable.*

Theorem 3: *If a language L is decidable then it is semidecidable.*

Theorem 4: *If a language L and is complement $\overline{L}$ are semidecidable then it is decidable.*

Theorem 5: *The class of semidecidable languages is not closed under complement.*

Theorem 6: *There exists languages that are not semidecidable.*

3 Not All Functions $f : \mathbb{N} \to \mathbb{N}$ are Computable

One of the primary uses of computing devices is to compute the value of functions. One obvious question is whether or not all functions can be computed? The answer to this is clearly no, which is probably no surprise to you. Especially since if we take a function from the reals to the reals the domain of the function is uncountable and we know that the number of Turing Machines, and hence algorithms, is countable. The restriction here is actually more prominent. Let's forget about all the problems with infinite decimal expansions or the real numbers and round-off error associated with computational devices and consider just functions from the natural numbers to the natural numbers, that is, $f : \mathbb{N} \to \mathbb{N}$. The natural numbers are a countable set, by definition. Even with these there are functions that cannot be computed. To show that something cannot be computed we simply need to establish that there is no Turing Machine that can compute it, hence there is no algorithm that can compute it.

We will do this in a couple different ways. One way is to proceed indirectly and utilize the material in Section 7 on the different sizes of infinity. If we consider the set of all of the functions $f : \mathbb{N} \to \mathbb{N}$. Each of these functions has a domain which is a subset of $\mathbb{N}$. Also, for any subset of $\mathbb{N}$ there is a function with that domain. So there are at least as many functions $f : \mathbb{N} \to \mathbb{N}$ as there are subsets of $\mathbb{N}$. The number of subsets of $\mathbb{N}$ is the power set of $\mathbb{N}$, denoted as $\mathcal{P}(\mathbb{N})$ or $2^{\mathbb{N}}$. From the results in Section 7 we know that $|\mathcal{P}(\mathbb{N})| > |\mathbb{N}|$, so $\mathcal{P}(\mathbb{N})$ is uncountable. Hence the number of functions $f : \mathbb{N} \to \mathbb{N}$ is uncountable and the number of Turing Machines (algorithms) is countable, hence there must exist functions on $\mathbb{N}$ that are not computable by any algorithm. In fact, along this line of reasoning there are actually an uncountably infinite number of functions on $\mathbb{N}$ that are not computable.

Another way to justify this is to essentially do Cantor's diagonalization proof that the

real number are uncountable.

Theorem 7: *There exists a function $f : \mathbb{N} \to \mathbb{N}$ that is not computable by any Turing Machine and hence any algorithm. Specifically, there is a function $f : \mathbb{N} \to \mathbb{N}$ that is not partial Turing computable.*

Proof. Every partial Turing computable function is computable by some Turing Machine. Let $\{T_k \mid T_k \text{ is a Turing Machine}\}$ be the set of all Turing Machines and let g_k be the function that is computed by the Turing Machine T_k, that is, $g_k(x) = T_k(x)$ for all $x \in Dom(g_k)$. Now define the function f as follows,

$$f(x) = \begin{cases} g_x(x) + 1 & \text{if } x \in Dom(g_x) \\ 0 & \text{otherwise} \end{cases}$$

We claim that the function f is not partial Turing computable. By Way of contradiction, assume that it is. Then there must exist some Turing Machine that computes it, that is, there is some machine T_i that computes f, so $f(x) = T_i(x)$ for all $x \in Dom(f)$. By the definition of f, we see that the domain of f is all of $\mathbb{N}$, that is $Dom(f) = \mathbb{N}$. So T_i must halt on the input i and furthermore $T_i(i) = f(i)$. But then $T_i(i) = f(i) = g_i(i) + 1 = T_i(i) + 1$ by the definitions of f and gx, a contradiction. Hence the function f is not partial Turing computable. That is, f cannot be computed by any algorithm. □

While this answers our question on computability, not everything can be computed by a mechanical computational device, it is a bit unsatisfying. While we constructed an uncomputable function, it was clearly a function of little interest. We know we can do computations like addition, multiplication, subtraction, division, modulus, powers and many higher-level functions like trigonometric and logarithmic functions as well as processes like differentiation and integration from Calculus to name just a few. It would be of much more interest if we could find something that was not computable and would also be useful. This is what the next two sections will be discussing.

4 The Halting Problem

Consider the following computational task,

> Given the code (that is text) of any program P, in any programming language, and any input X for the program, determine if P *halts* on X.

This would be a nifty addition to your debugging toolkit, would it not? It would determine if your program was sent into an infinite loop on some input. How do you determine this now? You run the program on lots of test data and if there is one scenario that causes the program to hang you *assume* that there is a problem and perhaps it is some never-ending loop. But there are many cases where this assumption would be wrong. Perhaps you gave it an input that simply takes a long time to process. For example, if you have a program

that factors integers into products of primes, some inputs will process quickly and some will take your computer years to process, even if the program is completely correct. If you had a program that would do the above task you would simply run it on your program and it would instantly tell you if you have an infinite loop or not, no guesswork and no incorrect assumptions.

The existence of such an algorithm is called the **Halting Problem**. That is, given the code of a program and input for the program determine whether or not the program halts on the input. As you probably guessed, there is no algorithm for accomplishing this task. If there was, and if it was sufficiently efficient you would have it in every debugging toolbox in every decent programming IDE.

Theorem 8: *There does not exist an algorithm for solving the Halting Problem. That is, there does not exist an algorithm such that, given the code of any program P and any input X for the program, that will determine if P halts on X.*

Proof. By way of contradiction, assume that an algorithm exists that will solve the Halting Problem. Then the algorithm can be implemented in the same programming language as the program P. Let H denote the implementation of this program. So H will take two inputs (parameters if you will) P and X, where P represents the code of the program and X the input data. Then, $H(P, X)$ will output 1 if P halts on X and 0 if P does not halt on X. Now create another program called D, where D is defined as $D(P) = H(P, P)$. In other words, $D(P)$ asks if the program P halts on its own code and returns either 0 or 1. Now create another program called C, where C is defined as follows. C takes a single input of the program code of P, calls the D program on P, if $D(P)$ returns 0 then C halts and if $D(P)$ returns 1, C goes into an infinite loop and does not halt. If the program H exists, then the programs D and C are easy to construct. Now let's take a look at the program C and run it on its own code. If $C(C)$ halts, then $D(C) = 0$, so $0 = D(C) = H(C, C)$ and by the definition of H this says that C does not halt on C, which is a contradiction. On the other hand, if $C(C)$ does not halt, then $D(C) = 1$, so $1 = D(C) = H(C, C)$ and by the definition of H this says that C halts on C, which is another contradiction. So C cannot halt on its own code nor can it not halt on its own code, therefore C cannot exist and thus our assumption that H exists is false as well. □

The above theorem shows that there is no algorithmic solution to the halting problem and as a consequence we cannot construct a program that will determine is another program will halt on a particular input.

5 Undecidability of Functional Properties

We are going to take this one step further. Although nonexistence of an algorithm to solve the halting problem is significant, there are other large families of non-computable tasks. We will look at one of them here.

In this section, as in the last, we will associate a program P with its code. So when we say a program P we are thinking of the what the program does and also the string of code that implements the program. If we think of a program P as its code we can also consider it as a word in a language since it would be an element of Σ^* over some alphabet Σ. So when we write $P \in L$ we are considering the code of P as a word in a language L.

Definition 9: *A program P has the* Property *defining the language L if $P \in L$ and does not have the property if $P \notin L$.*

Example 3: Say L is the language of all sorting programs. So if a program sorts a list it is in L and if the program does not sort a list then it is not in L. So if P is a program that does the quick-sort then $P \in L$ and if Q is a program that does a linear search of a list then $Q \notin L$.

Now let's consider two languages that define properties.

- $L_1 = \{P \mid P \text{ outputs 1 on an input of 0}\}$
- $L_2 = \{P \mid P \text{ halts on an input of 0 in at most two steps}\}$

The big difference between these two properties is that the first deals with *what* the program does and the second with *how* the program does it. We formalize this difference by defining a functional property.

Definition 10: *Let f be a function and let $\mathcal{P}_f$ be the set of all programs that compute f. A property L of programs is called* Functional *if*

1. *For any function f, either $P \in L$ for all programs $P \in \mathcal{P}_f$ or $P \notin L$ for all programs $P \in \mathcal{P}_f$.*
2. *Neither L nor $\overline{L}$ are empty.*

Example 4: Let $L_1 = \{P \mid P \text{ outputs 1 on an input of 0}\}$, then L_1 is a functional property. To prove that L_1 is a functional property, let f be any function. Then either $f(0) = 1$ or it does not. If $f(0) = 1$ then any program P that is in $\mathcal{P}_f$ will output 1 on an input of 0 and thus $P \in L_1$. On the other hand, if $f(0) \neq 1$ then any P that is in $\mathcal{P}_f$ will not output 1 on an input of 0 and thus $P \notin L_1$. Furthermore, there are many functions that output 1 on an input of 0 and those that don't as well as programs that implement them, hence neither L_1 nor $\overline{L_1}$ are empty.

Example 5: let $L_2 = \{P \mid P \text{ halts on an input of 0 in at most two steps}\}$, then L_2 is not a functional property. To see this we need a function that has some programs computing it that are in L_2 and some programs that compute it that are not in L_2. Consider the function $f(w) = w$, that is, the function that outputs the input. One program $P_1 \in \mathcal{P}_f$ simply does nothing and hence $P_1 \in L_2$. Another program $P_2 \in \mathcal{P}_f$ does three dummy actions that do nothing to the tape and then halts. So $P_2 \notin L_2$. Hence there are programs in $\mathcal{P}_f$ that are in L_2 and programs in $\mathcal{P}_f$ that are not in L_2, therefore L_2 is not a functional property.

Some other examples of functional properties,

- $\{P \mid P$ does not halt on an input of $1\}$
- $\{P \mid$ the number of inputs P halts on is finite$\}$
- $\{P \mid P$ halts on every input$\}$

Some examples of properties that are not functional,

- $\{P \mid P$ contains the transition $\delta(q_1, a_2) = (q_3, a_4, L)\}$
- $\{P \mid$ starting on an empty tape, P reaches state p_7 in at most five steps.$\}$

One of the most important results on functional properties is Rice's Theorem. We state Rice's Theorem without proof, a sketch to a proof can be found in [1].

Theorem 9: *(Rice's Theorem) Any functional property of programs is undecidable.*

In terms of programs and algorithms this means that if a language L defines a functional property there does not exist an algorithm that will decide if a program is in the language or not. So in the example of the language of sorting programs, a functional property, there would not be an algorithm that could determine if a program was in the language. More practically stated, there is no way to write a program that can take the code of another program and determine if it will sort a list or not.

6 Exercises

For each of the following properties determine if the property is functional or not. If it is functional prove it and if not find two programs in $\mathcal{P}_f$ one in L and the other not in L.

1. $L_1 = \{P \mid$ state q can be reached from state p in at most three steps$\}$
2. $L_2 = \{P \mid P$ halts in 10 or fewer steps on every input$\}$
3. $L_3 = \{P \mid P$ does not halt on an input of 2$\}$
4. $L_4 = \{P \mid P$ returns the square of the numeric input$\}$
5. $L_5 = \{P \mid$ there exists a configuration of P that yields a configuration with a given state $q\}$
6. $L_6 = \{P \mid P$ halts on all even number input$\}$
7. $L_7 = \{P \mid P$ is equivalent to a given program $Q\}$
8. $L_8 = \{P \mid P$ halts on no input$\}$

7 Notes on Infinity

7.1 Quick Review of some Discrete Mathematics

First a quick review of some topics from discrete mathematics.

Definition 11: *A function or map $f : X \to Y$ is an* injection (one-to-one) *if whenever $f(x_1) = f(x_2)$ then $x_1 = x_2$.*

Definition 12: *A function or map $f : X \to Y$ is a* surjection (onto) *if for every $y \in Y$ there exist at least one $x \in X$ such that $f(x) = y$.*

Definition 13: *A function or map $f : X \to Y$ is a* bijection (a one-to-one correspondence) *if f is both injective and surjective.*

Theorem 10: *A bijective function $f : X \to Y$ has an inverse function, denoted $f^{-1} : Y \to X$, which is also a bijection.*

Proof. Consult any text on discrete mathematics or logic, for example [3]. □

Example 6: Consider the two sets $\mathbb{N}$ and $E = \{2n \mid n \in \mathbb{N}\}$ and the function $f : \mathbb{N} \to E$ defined as $f(x) = 2x$. The function f is a bijection. To show that the function is injective we assume that there are two elements, x_1 and x_2, in the domain, $\mathbb{N}$, such that $f(x_1) = f(x_2)$ and we proceed to prove that $x_1 = x_2$. In many cases this reduces simply to some algebra,

$$\begin{aligned} f(x_1) &= f(x_2) \\ 2x_1 &= 2x_2 \\ x_1 &= x_2 \end{aligned}$$

Hence the function f is an injection. To show that the function is a surjection we take an arbitrary element, y of the codomain E and find an element x in the domain $\mathbb{N}$ such that $f(x) = y$. If y is an arbitrary element of E we know, by the definition of E, that $y = 2x$ for some $x \in \mathbb{N}$. So if we take the number x as the desired element we have $f(x) = 2x = y$, proving that the function is surjective and therefore a bijection.

7.2 Cardinality and Countability

Definition 14: *Two sets A and B have the same* cardinality *if there exists a bijective map f from A to B.*

Example 7: Consider the two sets $A = \{a, b, c\}$ and $B = \{1, 2, 3\}$. Clearly they have the same size (or cardinality), 3. But by our definition we must show the existence of a bijection f between them. Easy enough, let $f : A \to B$ be defined as $f(a) = 1$, $f(b) = 2$ and $f(c) = 3$. This is clearly a bijection and hence our sets have the same cardinality.

Notation: We denote the size, or cardinality of a set A by $|A|$. Note that if A is finite then $|A|$ is simply the number of elements in the set.

Example 8: Consider the two sets $\mathbb{N}$ and $E = \{2n \mid n \in \mathbb{N}\}$ from above and the function $f : \mathbb{N} \to E$ defined as $f(x) = 2x$. We have shown that this function is a bijection and hence E and $\mathbb{N}$ have the same cardinality, that is, they are the same size, $|\mathbb{N}| = |E|$. One nifty thing to note is that E is a proper subset of $\mathbb{N}$. So we have a proper subset that is the same size as the "bigger set". Some texts use this property to define an infinite set since this does not happen with finite sets. This also shows that the statement $A \subset B \Rightarrow |A| < |B|$ only applies to sets of finite size. Specifically, at least A being finite.

In our last example we used the notation $|A| < |B|$. What does this notation really mean? If the sets A and B are finite it is obvious. If A is finite then $|A| = n$ for some $n \in \mathbb{N}$ and if B is finite then $|B| = m$ for some $m \in \mathbb{N}$ and $|A| < |B|$ simply means that $n < m$. What if the sets A and B are infinite? Does it make any sense to write $|A| < |B|$? How can we say $\infty < \infty$? This last inequality is nonsense, of course. The main problem with the last inequality is the symbol ∞. It implies that there is one and only one "type" or "size" of infinity. This can not be any further from the truth as we will soon see.

This still leaves us with the problem of what $|A| < |B|$ means if both A and B are infinite sets. Here is a definition,

Definition 15: *Given two sets A and B we will say that $|A| < |B|$ if there exists an injection $f : A \to B$ but there does not exist a bijection between the two. Similarly, $|A| = |B|$ means that there does exist a bijection between the two sets and finally, $|A| \leq |B|$ means that there is either an injection of A into B or there is a bijection between the two sets.*

Now we will take a quick look at different sizes of infinity.

Definition 16: *A set is said to be* countable *(or* enumerable *or* denumerable*) if there is a bijective map from it to a subset of* $\mathbb{N}$*. A set that is not countable is said to be* uncountable*.*

Hence any finite set is countable, since if A is finite then $|A| = n$ and we can clearly create a bijective map from A to the set $\{1, 2, 3, \ldots, n\}$. One thing to note is that some textbooks define a countable set as being any set that can be put into a one-to-one correspondence with $\mathbb{N}$. If this were our definition then clearly finite sets would not be countable. We could use either definition for what we need to do so there is no reason to split hairs on this topic. I personally think that saying a finite set is not countable goes against our intuition since we can clearly count a finite set. There may be times when we wish to restrict our attention to just infinite sets that are countable, in which case we will say that the set is *countably infinite*.

Now let's do a few more examples, some will be obvious but others may surprise you.

Example 9: The set of all integers $\mathbb{Z}$ is countable. To show this we simply need to establish a bijection between $\mathbb{Z}$ and $\mathbb{N}$. To do this we can simply write down several specific input/output values until our pattern is established or we can work toward some closed form formula for the entire map. One way to get this map is as follows. We will let $f : \mathbb{N} \to \mathbb{Z}$ be the map and define f as $f(0) = 0$, $f(1) = -1$, $f(2) = 1$, $f(3) = -2$, $f(4) = 2$, ... For most audiences this is enough to establish the bijection. We could go one step further and write

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even.} \\ -(n+1)/2 & \text{if } n \text{ is odd.} \end{cases}$$

Our next example is a little surprising, at least I think so, and the map we will construct is a bit difficult to represent in closed form.

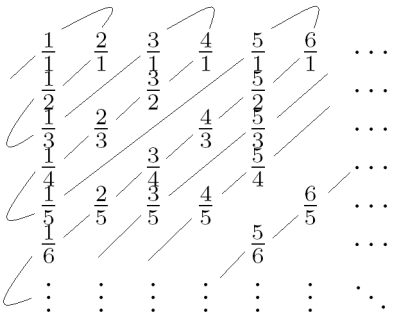
Example 10: The set of all rational numbers $\mathbb{Q}$ is countable. Recall that a rational number is any number that can be written in the form $\frac{a}{b}$ where $b \neq 0$. To show that this set is countable we will first show that the set of positive rational numbers is countable and then we can apply an argument similar to the one in the last example to get both the positive and negative rational numbers. So to show that the positive rational numbers are countable we need to establish a bijection from $\mathbb{N}$ to $\mathbb{Q}^+$. Here is the trick, write the set of rational numbers in a grid as follows.

$$\begin{array}{cccccccc}
\frac{1}{1} & \frac{2}{1} & \frac{3}{1} & \frac{4}{1} & \frac{5}{1} & \frac{6}{1} & \cdots \\
\frac{1}{2} & \frac{2}{2} & \frac{3}{2} & \frac{4}{2} & \frac{5}{2} & \frac{6}{2} & \cdots \\
\frac{1}{3} & \frac{2}{3} & \frac{3}{3} & \frac{4}{3} & \frac{5}{3} & \frac{6}{3} & \cdots \\
\frac{1}{4} & \frac{2}{4} & \frac{3}{4} & \frac{4}{4} & \frac{5}{4} & \frac{6}{4} & \cdots \\
\frac{1}{5} & \frac{2}{5} & \frac{3}{5} & \frac{4}{5} & \frac{5}{5} & \frac{6}{5} & \cdots \\
\frac{1}{6} & \frac{2}{6} & \frac{3}{6} & \frac{4}{6} & \frac{5}{6} & \frac{6}{6} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Before we proceed we need to convince ourselves that this list actually contains all of the positive rational numbers. It should be fairly clear since if someone were to hand you the rational number $\frac{n}{m}$ (with both n and m positive) you could go to the m^{th} row and the n^{th} column and $\frac{n}{m}$ would be in that position. Now since a bijection is an injection we need to remove duplicates from the grid. Note that $\frac{1}{1} = \frac{2}{2} = \frac{3}{3} = \cdots$, $\frac{1}{2} = \frac{2}{4} = \frac{3}{6} = \cdots$. So we will simply go through the grid and remove any number that has been previously encountered. Another way to interpret this action is removing any fraction that is not in lowest terms.

$$\begin{array}{cccccccc}
\frac{1}{1} & \frac{2}{1} & \frac{3}{1} & \frac{4}{1} & \frac{5}{1} & \frac{6}{1} & \cdots \\
\frac{1}{2} & & \frac{3}{2} & & \frac{5}{2} & & \cdots \\
\frac{1}{3} & \frac{2}{3} & & \frac{4}{3} & \frac{5}{3} & & \cdots \\
\frac{1}{4} & & \frac{3}{4} & & \frac{5}{4} & & \cdots \\
\frac{1}{5} & \frac{2}{5} & \frac{3}{5} & \frac{4}{5} & & \frac{6}{5} & \cdots \\
\frac{1}{6} & & & & \frac{5}{6} & & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Now to establish the map we zig-zag through the grid diagonally.



So our map would look something like this, $f(0) = 1$, $f(1) = 2$, $f(2) = \frac{1}{2}$, $f(3) = \frac{1}{3}$, $f(4) = 3$, $f(5) = 4$, $f(6) = \frac{3}{2}$, $f(7) = \frac{2}{3}$, $f(8) = \frac{1}{4}$, ... To finish the example we apply the

last example's method to get the negative numbers into the map. For ecample, $f(0) = 0$, $f(1) = -1$, $f(2) = 1$, $f(3) = -2$, $f(4) = 2$, $f(5) = -\frac{1}{2}$, $f(6) = \frac{1}{2}$, $f(7) = -\frac{1}{3}$, $f(8) = \frac{1}{3}$, $f(9) = -3$, $f(10) = 3$, $f(11) = -4$, $f(12) = 4$, $f(13) = -\frac{3}{2}$, $f(14) = \frac{3}{2}$, $f(15) = -\frac{2}{3}$, $f(16) = \frac{2}{3}$, $f(17) = -\frac{1}{4}$, $f(18) = \frac{1}{4}$, ...

This zig-zag method can be used to prove some other interesting facts about countable sets.

Theorem 11: *The union of a countable collection of countable sets is countable.*

Proof. Let A be the countable set of countable sets, so $A = \{A_1, A_2, A_3, \ldots\}$ with $A_1 = \{a_{1,1}, a_{1,2}, a_{1,3}, \ldots\}$, $A_2 = \{a_{2,1}, a_{2,2}, a_{2,3}, \ldots\}$, $A_3 = \{a_{3,1}, a_{3,2}, a_{3,3}, \ldots\}$, and so on. Since each of the A_i are countable and A is a countable collection of the A_i we can write the set of all elements of all the A_i into a single grid as,

$$\begin{array}{ccccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & \ldots \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & \ldots \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & \ldots \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

Now use the same zig-zag correspondence we used to show that the rational numbers were countable. In other words, order the union of elements as

$$\{a_{1,1}, a_{1,2}, a_{2,1}, a_{3,1}, a_{2,2}, a_{1,3}, a_{1,4}, a_{2,3}, a_{3,2}, a_{4,1}, \ldots\}$$

and use this ordering as the bijection with $\mathbb{N}$. □

Now let's look at an example of a set that is not countable.

Example 11: The set of real numbers, $\mathbb{R}$, is uncountable. The proof we will use is called Cantor's Diagonalization Proof. It is a proof by contradiction. Before starting the proof recall that we can define the set of real numbers as the set of all decimal expansions. What we will show is that the set of real numbers between 0 and 1 is uncountable. That is, the set of numbers in the interval $[0, 1]$. If we establish that this set is uncountable then it clearly follows that the entire set of real numbers is uncountable. By way of contradiction we will assume that the set of real numbers in the interval $[0, 1]$ is countable. This implies that there is a one-to-one correspondence between $\mathbb{N}$ and $[0, 1]$. If there is then we can list all of the real numbers in $[0, 1]$ like so,

$$\begin{array}{l} 0.a_{1,1}\ a_{1,2}\ a_{1,3}\ a_{1,4}\ a_{1,5}\ a_{1,6}\ldots \\ 0.a_{2,1}\ a_{2,2}\ a_{2,3}\ a_{2,4}\ a_{2,5}\ a_{2,6}\ldots \\ 0.a_{3,1}\ a_{3,2}\ a_{3,3}\ a_{3,4}\ a_{3,5}\ a_{3,6}\ldots \\ 0.a_{4,1}\ a_{4,2}\ a_{4,3}\ a_{4,4}\ a_{4,5}\ a_{4,6}\ldots \\ 0.a_{5,1}\ a_{5,2}\ a_{5,3}\ a_{5,4}\ a_{5,5}\ a_{5,6}\ldots \\ 0.a_{6,1}\ a_{6,2}\ a_{6,3}\ a_{6,4}\ a_{6,5}\ a_{6,6}\ldots \\ \vdots \end{array}$$

The important thing to remember is that because of the assumption of countability we are guaranteed that *all* of the real numbers in $[0, 1]$ are in the above list. So to show that this set is uncountable we will simply find a real number in $[0, 1]$ that is not in this list. As for the notation in the above list, $a_{i,j}$ is the j^{th} decimal place of the i^{th} number. Now consider the diagonal of this grid, that is, the $a_{i,i}$ entries.

$$
\begin{array}{l}
0.a_{1,1}\ a_{1,2}\ a_{1,3}\ a_{1,4}\ a_{1,5}\ a_{1,6}\ldots \\
0.a_{2,1}\ a_{2,2}\ a_{2,3}\ a_{2,4}\ a_{2,5}\ a_{2,6}\ldots \\
0.a_{3,1}\ a_{3,2}\ a_{3,3}\ a_{3,4}\ a_{3,5}\ a_{3,6}\ldots \\
0.a_{4,1}\ a_{4,2}\ a_{4,3}\ a_{4,4}\ a_{4,5}\ a_{4,6}\ldots \\
0.a_{5,1}\ a_{5,2}\ a_{5,3}\ a_{5,4}\ a_{5,5}\ a_{5,6}\ldots \\
0.a_{6,1}\ a_{6,2}\ a_{6,3}\ a_{6,4}\ a_{6,5}\ a_{6,6}\ldots
\end{array}
$$

For each i select a number $b_i \neq a_{i,i}$ and construct the number $B = 0.b_1b_2b_3b_4b_5b_6\ldots$. We claim that B is not in the list we first constructed. B can not be equal to the first number in the list since the first decimal place of B differs from the first decimal place of $0.a_{1,1}a_{1,2}a_{1,3}a_{1,4}a_{1,5}a_{1,6}\ldots$. B can not be equal to the second number in the list since the second decimal place of B differs from the second decimal place of $0.a_{2,1}a_{2,2}a_{2,3}a_{2,4}a_{2,5}a_{2,6}\ldots$. B can not be equal to the third number in the list since the third decimal place of B differs from the third decimal place of $0.a_{3,1}a_{3,2}a_{3,3}a_{3,4}a_{3,5}a_{3,6}\ldots$, and so on. Hence B is a real number in the interval $[0, 1]$ that is not in our original list. Hence we can not list all real numbers in the interval $[0, 1]$ and therefore the set of real numbers in the interval $[0, 1]$ is uncountable. As we mentioned above, this implies that the set of real numbers is uncountable.

If the above argument is too abstract consider this illustrative sub example, the list would look something like the following.

$$
\begin{array}{l}
0.1248273994832\ldots \\
0.2539188773628\ldots \\
0.9923019928383\ldots \\
0.1029938847722\ldots \\
0.6472837462827\ldots \\
0.5559828728393\ldots \\
0.2847284950050\ldots \\
\vdots
\end{array}
$$

and the number B would be something like, $B = 0.4718952\ldots$

Note what this says about irrational numbers like π, e, and $\sqrt{2}$. If we let I represent the set of irrational numbers we know that $\mathbb{R} = \mathbb{Q} \cup I$. If I were countable then $\mathbb{R}$ would be a union of two countable sets and hence be countable, which we know is not the case. Therefore the set of irrational numbers is uncountable. Thus there exists far more irrational numbers than rational numbers.

7.3 An Infinite Progression of Cardinals

The material in the previous section shows us that there are at least two different sizes of infinity, countable and uncountable. One can only ask is this the end of the story or is there more here. From our definition of cardinality it is clear that all countable sets have the same cardinality but what about the uncountable sets. All we know is that there is not a bijection between an uncountable set and $\mathbb{N}$, but that is all we know.

Recall from discrete mathematics the concept of the power set of a set. The power set of a set, A, is the set of all subsets of A.

Example 12: Let $A = \{a, b, c\}$ then the power set of A is

$$\mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}\}$$

As you know from discrete mathematics if A is finite, $|A| = n$ then the order of the power set of A is $|\mathcal{P}(A)| = 2^n$. So for finite sets it is obvious that A and $\mathcal{P}(A)$ have different cardinalities. Does the same hold for infinite sets? This question is a little harder to answer since both A and $\mathcal{P}(A)$ are infinite. To verify a difference in cardinalities we need to show that there does not exist a bijection between the two sets.

Theorem 12: $|A| < |\mathcal{P}(A)|$

Proof. Since there is an injection $g : A \to \mathcal{P}(A)$ defined by $g(a) = \{a\}$ we have $|A| \leq |\mathcal{P}(A)|$. By way of contradiction assume that $|A| = |\mathcal{P}(A)|$, then there exists a bijection $f : A \to \mathcal{P}(A)$. Consider the set $A' = \{a \in A \mid a \notin f(a)\}$. Since $A' \in \mathcal{P}(A)$ there exists $a' \in A$ with $f(a') = A'$. Now either $a' \in A'$ or $a' \notin A'$. If $a' \in A'$ then by the definition of A', $a' \notin f(a') = A'$, a contradiction. If $a' \notin A'$ then again by the definition of A', $a' \in f(a') = A'$, a contradiction. Thus no bijection f exists and $|A| < |\mathcal{P}(A)|$. □

This gives us an infinite sequence of progressively larger sizes of infinity.

$$|\mathbb{N}| < |\mathcal{P}(\mathbb{N})| < |\mathcal{P}(\mathcal{P}(\mathbb{N}))| < |\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))| < \cdots$$

Note that each of the sets $\mathcal{P}(\mathbb{N}), \mathcal{P}(\mathcal{P}(\mathbb{N})), \mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N}))), \ldots$ is uncountable. So there are an infinite number if sizes of infinity that are all uncountable.

One obvious question that arises from the above list is whether or not this list gets all of the sizes of infinity. That really breaks into two separate questions. First, is there a size of infinity that is larger than any of those in the above list? And is there a size of infinity that is between any of these?

The answer to the first questions is, yes. There are sizes of infinity larger than any that can be formed by $\mathcal{P}(\mathcal{P}(\cdots \mathcal{P}(\mathbb{N}) \cdots))$. The answer to the second question is still unknown. To simplify the discussion we will introduce some new notation.

Notation: We denote $|\mathbb{N}| = \aleph_0$ and $|\mathbb{R}| = \aleph$. These numbers are referred to as infinite cardinals or infinite cardinal numbers.

Definition 17: *We define the sequence of infinite cardinals* $\aleph_0$, $\aleph_1$, $\aleph_2$, ... *to mean that* $\aleph_i < \aleph_{i+1}$ *for all* i *and that there is no infinite cardinal* $\aleph'$ *such that* $\aleph_i < \aleph' < \aleph_{i+1}$ *for any* i. *In terms of sets this means that for any* i *there exists a set* A_i *such that* $|A_i| = \aleph_i$. *Also, there exists an injection from* A_i *to* A_{i+1} *for all* i *and there does not exist a bijection from* A_i *to* A_{i+1}. *Furthermore, it also means that there does not exist a set* B *such that there are injections from* A_i *to* B *and* B *to* A_{i+1} *and no bijections between* A_i *and* B *or* B *and* A_{i+1}.

Notation: Some logic texts will use exponential notation to denote the infinite cardinals. For example, $|\mathbb{N}| = \aleph_0$, $|\mathcal{P}(\mathbb{N})| = 2^{\aleph_0}$, $|\mathcal{P}(\mathcal{P}(\mathbb{N}))| = 2^{2^{\aleph_0}}$, $|\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))| = 2^{2^{2^{\aleph_0}}}$, ... In general, if a set A has size $|A| = p$, where p could be finite or infinite then the size of its power set is denoted $|\mathcal{P}(A)| = 2^p$.

Now we know that $|\mathbb{N}| < |\mathcal{P}(\mathbb{N})| < |\mathcal{P}(\mathcal{P}(\mathbb{N}))| < |\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))| < \cdots$ and $|\mathbb{N}| = \aleph_0$. We also know, although we have not proven it, that $|\mathcal{P}(\mathbb{N})| = |\mathbb{R}| = \aleph$. One of our questions above can be stated as, does $\aleph_1 = \aleph$? That is, is $|\mathbb{R}|$ the next infinite cardinal? This question is still unknown but from the work that has been done so far it looks promising that this is indeed the case. This is known as the continuum hypothesis, note it is a hypothesis not a theorem since it has not been proven. Specifically, the continuum hypothesis states that $|\mathbb{R}| = \aleph_1$. That is, $|\mathcal{P}(\mathbb{N})| = \aleph_1$, so there is no different size of infinity between $|\mathbb{N}| = \aleph_0$ and $|\mathcal{P}(\mathbb{N})| = \aleph_1$. The generalized continuum hypothesis states that $2^{\aleph_i} = \aleph_{i+1}$ for all i. This is equivalent to saying that the sequence of infinite cardinals $\aleph_0$, $\aleph_1$, $\aleph_2$, $\aleph_3$, ... is $|\mathbb{N}|$, $|\mathcal{P}(\mathbb{N})|$, $|\mathcal{P}(\mathcal{P}(\mathbb{N}))|$, $|\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))|$, ... So if we assume that the generalized continuum hypothesis is true then our sequence

$$|\mathbb{N}| < |\mathcal{P}(\mathbb{N})| < |\mathcal{P}(\mathcal{P}(\mathbb{N}))| < |\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))| < \cdots$$

has no sizes of infinity in between those displayed in the sequence. Furthermore we have examples of specific sets that have sizes $\aleph_0$, $\aleph_1$, $\aleph_2$, $\aleph_3$, ...

This still does not answer the question of whether or not

$$|\mathbb{N}|, |\mathcal{P}(\mathbb{N})|, |\mathcal{P}(\mathcal{P}(\mathbb{N}))|, |\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))|, \ldots$$

is all of them. There is still a chance that there exists a set whose size is larger than all of these. That is, there could exist a set B in which there is an injection from A_i to B for all i and yet no bijection between B and any of the A_i. Our next goal is to prove that there is such a set B. The proof of the existence of such a set relies on the Axion of Choice. Notice that this is an axiom, not a theorem. That means that we either accept it or we do not. Most mathematicians accept the Axiom of Choice without hesitation but there are some who refuse to. We will, of course, accept it. Here are some equivalent statements of the axiom of choice. These and other equivalent statements can be found in [3] and [4].

Axiom of Choice 1: If $\mathcal{A}$ is a disjoint collection of nonempty sets, then there exists a set B such that for each A in $\mathcal{A}$, $B \cap A$ is a unit set, that is, contains a single element.

Axiom of Choice 2: For every set X there exists a function f on the collection, $\mathcal{P}(X) - \{\emptyset\}$, of nonempty subsets of X such that $f(A) \in A$.

The function f in the above statement of the axiom of choice is sometimes called a choice function because it reaches into a set and chooses an element from that set.

Axiom of Choice 3: If $\{A_i\}$ is a family of nonempty sets indexed by a nonempty set I, then $\mathrm{X}_{i\in I}A_i$ is nonempty. Where $\mathrm{X}_{i\in I}A_i$ represents the (possibly infinite) cartesian product of the A_i.

Now the existence of arbitrarily large cardinal numbers.

Theorem 13: *If $\mathcal{C}$ is a set of cardinal numbers, then there exists a cardinal number greater then each cardinal in $\mathcal{C}$.*

Proof. Since each cardinal number is associated with a set of that size we can consider $\mathcal{C}$ to be a disjoint collection of these sets. Using the axiom of choice we can define a representative set of the form $\mathcal{A} = \{A_u \mid u \in \mathcal{C}\}$, where $|A_u| = u$. Clearly, $|\cup_{u\in\mathcal{C}} A_u| \geq u$ for each $u \in \mathcal{C}$. Hence $|\mathcal{P}(\cup_{u\in\mathcal{C}}A_u)| = 2^{|\cup_{u\in\mathcal{C}}A_u|} > |\cup_{u\in\mathcal{C}} A_u|$, and so $|\mathcal{P}(\cup_{u\in\mathcal{C}}A_u)|$ is a cardinal number that exceeds all of the cardinal numbers in $\mathcal{C}$. □

So the upshot is that $|\mathbb{N}|$, $|\mathcal{P}(\mathbb{N})|$, $|\mathcal{P}(\mathcal{P}(\mathbb{N}))|$, $|\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))|$, ... is not all of them. We have,

$$\begin{aligned}|\mathbb{N}| < |\mathcal{P}(\mathbb{N})| < |\mathcal{P}(\mathcal{P}(\mathbb{N}))| < |\mathcal{P}(\mathcal{P}(\mathcal{P}(\mathbb{N})))| < \cdots \\ < |\mathcal{P}(\cup_{u\in\mathcal{C}}A_u)| < |\mathcal{P}(\mathcal{P}(\cup_{u\in\mathcal{C}}A_u))| < \cdots \\ < p < 2^p < 2^{2^p} < \cdots < q < 2^q < 2^{2^q} < \cdots\end{aligned}$$

where p is some cardinal larger than $|\cup\mathcal{P}(\cdots\mathcal{P}(\cup_{u\in\mathcal{C}}A_u)\cdots)|$ for example the size of the power set of that union, $|\mathcal{P}(\cup\mathcal{P}(\cdots\mathcal{P}(\cup_{u\in\mathcal{C}}A_u)\cdots))|$ and q is some cardinal larger than $|A_p \cup A_{2^p} \cup A_{2^{2^p}} \cup \cdots|$ for example $|\mathcal{P}(A_p \cup A_{2^p} \cup A_{2^{2^p}} \cup \cdots)|$. And this is not by any means an exhaustive list, at least as far as we know.

References

[1] *Theory of Computing: A Gentle Introduction*, by Efim Kimber and Carl Smith, Prentice Hall, 2001.

[2] *An Introduction to Formal Languages and Automata*, Fifth Edition, by Peter Linz, Jones and Bartlett, 2012.

[3] *Set Theory and Logic*, by Robert R. Stoll, W. H. Freeman and Company, 1963.

[4] *The Foundations of Mathematics*, by Raymond L. Wilder, John Wiley & Sons, 1952.