Decidability

Frank

January 12, 2022

1 Paradoxes

1.1 Barber's Paradox

- Village has a male barber who shaves every man in the village who does not shave himself.
- Question: Does the barber shave himself?

  If yes, then no

  If not, then yes

  This means that Yes $\iff$ No, which contradicts the existence of such a village

  Self-reference with negation turns out to be problematic

1.2 Liar's Paradox (Epimenides' Paradox)

- Consider the statement: "This statement is false"
- Again we see a self-reference with negation

  If true, then false

  If false, then true

  This means True $\iff$ False, which is another paradox

  This can be due to the fact that our language is so powerful, that we can create statements that are paradoxical, but can we restrict our language?

1.3 Russel's Paradox

- $R =$ the set of all sets that are not members of themselves

  $= \{X : X \text{ is a set and } X \notin X\}$

  Ex. The empty set is not a member of itself, because the empty set has no elements. $\emptyset \neq \{\emptyset\}$, but then it is a member of itself, because every set contains the empty set. If it is a member of itself, but it is empty, so we have a contradiction.

  However, the set of all sets is a member of itself, because it is the set of all sets

- Does this set contain itself ($R \in R$)?

  Yes, then $R \notin R$ by definition of this set

  No, then $R \in R$, by definition of this set.

  Then $R \in R \iff R \notin R$

1.4 Godel's Incompleteness Theorem (Not a paradox)

- Using elementary mathematics involving just $\mathbb{Z}, +, \times$, he wrote a mathematical statement S whose meaning is

  $S =$"This statement is unprovable"

  If S is true, then S is true unprovable statement

  If S is false, then S is false provable statement

  Neither options are desirable. First being our mathematics involving the operations above cannot be used to prove all true statements, and we definitely do not want to prove false statements (we want to prove statements to be true).

2 Sizes of Infinite Sets

2.1 Galileo (1638)

- Consider $\mathbb{N} : 0, 1, 2, 3, \ldots$

  Perfect squares: $0, 1, 4, 9, \ldots$

  Then these 2 sets have the same size, since we have a 1-1 correspondence between the naturals and the perfect squares

2.2 Cantor (1874)

- **Def**: Sets $A\&B$ are equinumerous (same size and cardinality). ($|A| = |B|$)
  iff $\exists$ bijection (1-1 and onto) $f : A \to B$ (1-1 correspondence)
- **Def**: A is countable (or enumerable) iff A is finite or equinumerous to $\mathbb{N}$ (countably infinite. ex. $f : \mathbb{N} \to A$)
- **Enumeration of a countable set** is a sequence $a_0, a_1, a_2, \ldots$ so that each element of A appears exactly (but not necessary) once in some particular index (represented by a natural) of this list ($f(n) = a_n$). Then we have a bijection.

2.3 Examples - Countable

- **Thm 1.1**: $\mathbb{Z}$ is countable

  $\mathbb{Z} = \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$

  $0, 1, 2, 3, \ldots, -1, -2, -3, \ldots$

  Are not enumerations because we cannot determine an exact index in which we can find a certain element in the sequence (infinite on both sides)

- Consider $0, 1, 2, 3, 4, \ldots$ and $-1, -2, -3, -4, \ldots$

  Then we can construct $0, -1, 1, -2, 2, -3, 3, \ldots$, which is an enumeration of $\mathbb{Z}$ (Given an element, we can identify where that element will be in the sequence) - This is called "dovetailing" (Taking numbers from multiple lists and merging them into a single list)

  Note that we can put a 1-1 correspondence between the sequence $0, 1, 2, 3, \ldots$ (set of naturals) and the sequence $0, -1, 1, -2, \ldots$ (enumeration of integers)

- **Thm 1.2**: $\mathbb{N} \times \mathbb{N}$ is countable

  $\mathbb{N} \times \mathbb{N} =$

  $$(0,0), (0,1), (0,2), \dots$$
  $$(1,0), (1,1), (1,2), \dots$$
  $$(2,0), (2,1), (2,2), \dots$$
  $$\dots\dots\dots$$

  We can enumerate this array diagonally (look at set of pairs of natural numbers whose sum is $0, 1, 2, 3, \dots$ in order)

  For example, look at $(0,0), (1,0), (0,1), (2,0), (1,1), (0,2), \dots$

  First list all pairs (i,j) s.t. $i+j=0$ [1]
  Then "" (i,j) s.t. $i+j=1$ [2]
  Then "" $i+j=2$ [3]

- Check that (i,j) appears in position $\frac{(i+j)(i+j+1)}{2} + i$

  If we visit pairs in diagonals pointing downwards, then the above formula would be correct

  Then we have defined a pairing function: $\tau_2(i,j) = \frac{(i+j)(i+j+1)}{2} + i$ (bijection), which encodes pairs of natural numbers by one natural number (1-1 correspondence with naturals)

  Then $\tau_2^{-1} : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ decodes the pair that corresponds to a position in the enumeration.

- **Exercise**: Show that $\mathbb{Z} \times \mathbb{Z}$ is countable

- **Cantor**: $\mathbb{Q}^+ = \{\frac{a}{b} : a, b \text{ are pos integers}\}$ is countable (dense set, but countable)

  Can do

  $$\begin{array}{cccc} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \cdots \\ \frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \frac{2}{4} \cdots \\ \frac{3}{1} & \frac{3}{2} & \frac{3}{3} & \frac{3}{4} \cdots \\ \frac{4}{1} & \frac{4}{2} & \frac{4}{3} & \frac{4}{4} \cdots \\ \cdots\cdots & & & \end{array}$$

  and apply the diagonal (snake) technique to enumerate. Note that this enumeration has many repeats for the same element. In fact, we repeat the same element an infinite amount of times (ex. $\frac{1}{2}$ and $\frac{2}{4}$). However, just skip these elements in the enumeration so we still have a bijection.

- **Fact**: $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ is countable too.

  Trick: $(i,j,k) \sim ((i,j),k)$, then $\tau_3(i,j,k) = \tau_2(\tau_2(i,j),k)$, which is also a bijection

  $\tau_3 : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

- **Thm 1.3**: $\forall k \in \mathbb{N}, \mathbb{N}^k = \mathbb{N} \times \mathbb{N} \times \ldots \times \mathbb{N}$ (k-times) is countable

  Proof: Use induction

- **Exercise**: The set of all finite sequences of natural numbers is countable

2.4 Examples - Uncountable

- **Thm 1.4**: The set B^∞ of infinite binary strings is not countable

  proof (diagonal argument): assume for contra, that B^∞ is countable.

  Let $x_0, x_1, x_2, \ldots$ be any enumeration of B^∞

  Ex.

$$\begin{aligned} x_0 &= 01101011\ldots \\ x_1 &= 00110101\ldots \\ x_2 &= 11101011\ldots \\ x_3 &= \ldots \\ &\vdots \end{aligned}$$

  Let b_{ij} = j-th bit of x_i

  Consider the diagonal $b_{00}, b_{11}, \ldots$

  Construct infinite binary string x whose i-th bit is $\bar{b}_{ii}$ (complement), then $x = 110\ldots$

  Then note that x does not appear (cannot appear) in our enumeration by construction

  x differs from x_i in (at least) the i-th position, so $x \neq x_i$ for all $i \in \mathbb{N}$

  Thus, $x_0, x_1, x_2, \ldots$ is not an enumeration of B^∞. Thus we have a contradiction.

  $\therefore B^\infty$ is not countable

- **Cantor**: The set of reals in $(0, 1]$ is not countable!

  Example

$$\begin{aligned} r_0 &= .93625\ldots \\ r_1 &= .83720\ldots \\ r_2 &= .62193\ldots \\ &\vdots \end{aligned}$$

  By the argument above, we can also take the diagonal $(931\ldots)$ and construct a number that does not appear anywhere in the enumeration (if we have 9, pick something other than 9, etc.)

- **Thm 1.5**: The set of functions $\mathbb{N} \to \mathbb{N}$ is not countable

  proof: By diagonal argument

  Assume the set is countable, consider an arbitrary enumeration

  Diagonalization

	0	1	2	...
f_0	17	3	0	
f_1	0	13	78	
f_2	0	0	0	
...				

  Now create a function f s.t. $f(i) \neq f_i(i)$ for all $i \in \mathbb{N}$

  For example, $f(i) = f_i(i) + 1$, then f does not appear in the enumeration, which leads to a contradiction.

  $\therefore$ the set of functions from $\mathbb{N} \to \mathbb{N}$ is uncountable

2.5 Another Countable Example

- Consider Γ = [finite] alphabet (set of symbols), $|\Gamma| \geq 2$

  Then $\Gamma^\star$ = set of finite strings over Γ

  Thm 1.6: $\Gamma^\star$ is countable

  Intuitively, we have a finite number of strings for each string of length k (In fact, for string of length k, we can have $|\Gamma|^k$ strings), thus we can create an assignment from each natural to a unique string.

  proof: Enumerate all strings over Γ of length $0, 1, 2, 3, \ldots$ in lexicographic order

3 Uncomputability (Informally)

3.1 Introduction

- There are things that computers cannot do but are supposed to!
- Note that computers take an input, and produce an output, in binary. Thus, computers compute functions $\mathbb{N} \to \mathbb{N}$

  There are functions $\mathbb{N} \to \mathbb{N}$ that computers cannot compute

- **Cheap Argument** (Cheap counting argument): There is an uncountable number of functions $\mathbb{N} \to \mathbb{N}$, but only a countable number of programs to compute them. This is because every computer program is made out of a finite number of characters over an alphabet, thus we have a finite number of programs we can write. (Compiler must stop at some point, thus we cannot have an infinite number of characters)

3.2 Turing's Halting Function

- Function:

$$h(P,x) = \begin{cases} 1 & \text{if program } P \text{ halts on input } x \\ 0 & \text{if } P \text{ does not halt on } x \end{cases}$$

Note that since a program is a string of bits, then P is a number, and x is also a number

However, we have a pair of numbers here. But then again, we can encode the pairs with a single number that encode the pair, so $h(P,x)$ is really just another function $\mathbb{N} \to \mathbb{N}$

- **Fact (1936)**: No program can compute h

Note that if just run the program P and it halts, then we're fine, but if it doesn't halt, how will we know that the program never halts? So we need a program that looks at the input and knows that it will or will not halt.

proof: Suppose for contra, that such a program exists, call it H (H returns the appropriate values based on whether $P(x)$ halts or not), then

$$H(P,x) = \begin{cases} \text{return } 1 & \text{if program } P(x) \text{ halts} \\ \text{return } 0 & \text{if } P(x) \text{ does not halt} \end{cases}$$

Modify H to obtain H': replace every "return b" statement in H by

"if $b = 1$ then

while true do $x \leftarrow x$ (infinite loop)

else return 0"

Then we end up with the function

$$H'(P,x) = \begin{cases} \text{does not halt}, & \text{if } P(x) \text{ halts} \\ \text{return 0 (halts)} & \text{if } P(x) \text{ does not halt} \end{cases}$$

Now write $DG(P)$

$DG(P)$: return $H'(P,P)$, then

$$DG(P) = \begin{cases} \text{does not halt}, & \text{if } P(P) \text{ halts} \\ \text{return 0 (halts)} & \text{if } P(P) \text{ does not halt} \end{cases}$$

But since DG is another program, we can run DG on input DG:

$$DG(DG) = \begin{cases} \text{does not halt}, & \text{if } DG(DG) \text{ halts} \\ \text{return 0 (halts)} & \text{if } DG(DG) \text{ does not halt} \end{cases}$$

Then we have a contradiction, $DG(DG)$ halts $\iff$ $DG(DG)$ does not halt

Reconsidering our assumption H, then H does not compute h correctly, since we have arrived at a contradiction by manipulating H. All programs that attempt to compute h is subjected to such an argument. Thus, we cannot compute h

But how does this relate to diagonalization? (the function name DG)

Consider $H'(P,P)$, the tuple (P,P) is our diagonal

- Consider the enumeration of the program H, any program P_i against the inputs on the program (Note that these are all numbers). Then the program H tells whether the inputs $0, 1, 2, \ldots$ makes each program P_i halt.

	0	1	2	...
P_0	0	1	1	
P_1	1	0	0	
P_2	0	0	1	
...				

Note that for program H, we have $H(i, j) = 1$ if P_i halts on j, 0 if P_i doesn't halt on j.

In our program H', look at what happens on the diagonal, if P_i halts on i then it doesn't halt, if P_i doesn't halt on i, then it halts, which gives us the contradiction. (Note that only diagonal applies, because we do not have a reciprocal nature, i.e. P_i halts on j and P_j halts on i, which does not give a contradiction).

This means on the diagonal, we have the special property that "if $P_i(i)$ halts, then it doesn't halt, and if $P_i(i)$ doesn't halt, then it halts", which is a contradiction.

4 Reductions

4.1 Introduction

- A problem X reduces to problem Y, $X \leq Y$, if we can use a (given or assumed) solution to Y to solve X

  $X \leq Y$ means X is no harder than Y (Positive Use)

  Y is at least as hard as X (Negative Use)

- **Positive Use**: Know how to solve Y but not X and X reduces to Y, then can just solve Y to indirectly solve X

- **Negative Use**: Know that X is hard and that X reduces to Y, then Y is hard, since Y is at least as hard as X

4.2 Example

- Consider the problem ZERO: Given program P and input x, determine if $P(x)$ returns 0.

  Halt: Halting Problem.

  Want that HALT $\leq$ ZERO, so that we can prove ZERO is not computable

  Suppose Z-SOLVER is a program that solves ZERO

$$\text{Z-SOLVER(P, x)} = \begin{cases} 1 & \text{if } P(x) = 0 \\ 0 & \text{otherwise} \end{cases}$$

  Reduce HALT to ZERO by using Z-SOLVER to solve HALT

- Consider H-SOLVER(P, x):

  $P' \leftarrow$ program obtained from P by changing every "return x" to "return 0"
  return Z-SOLVER(P', x)

$$\text{H-SOLVER(P, x)} = \begin{cases} 1 \text{ (halts)} & \text{if } P'(x) \text{ returns 0 } (P(x) \text{ halts}) \\ 0 & \text{if } P'(x) \text{ does not return 0 } (P(x) \text{ does not halt}) \end{cases}$$

So therefore, H-SOLVER solves the halting problem, which is not computable, so Z-SOLVER is not computable, since HALT $\leq$ ZERO