Infinite Computable

Not a new concept

What does infinite mean?

- The concept of infinity exists in our minds
- Does **infinity** actually exist?
- What does **actually** mean?

Is it a physical embodiment?

Between any two **locations** in space there is a third location?

Or after every **moment** in time there is a next moment?

Cogito

- As a concept, no doubt it (infinity) exists

This is why it was given a name in the first place

- The same applies for:

circle

straight line

Even a point

Almost everything

Can computers handle infinity?

- What do you mean by handle?
- What do you mean by computers?
- Alright, computers are Turing Machines

Can Turing Machines handle infinity?

- Still, what do you mean by handle?

- Do you mean save (encode) the whole set in the TM's memory?

- Well, they have infinite tapes.

Can a **PHYSICAL** computer handle ∞?

Can a physical computer handle an infinite set?

- If you mean save all of it, then according the physics we know so far, NO
- We don’t need to save a whole set
- Computability is about answering membership questions

Example

- The set of numbers divisible by 5 exists as a concept

- You can write a program, which can work for any given natural input, to tell us if the given number is divisible by 5 or not.

Simply, look at the first digit from the right and check if it is 0 or 5.

Did we forget something?

- What does it mean to be **given** a number
- What if the number is too large to be saved as an input
- Might take forever to find where it starts

TM’s are the best

- Tell me about a better way to talk about computers with arbitrary capacity

- TM’s for physical computers, are like the Circle for the sun

- The first is an ideal concept which smoothens a physical entity

- Or maybe the second is a rough physical manifestation of an ideal reality

Final words on infinite sets

- Handling infinity is problematic regardless of the whole computers talk

- Even problematic regardless of any physical realizations

- Check The axiom of choice

Sets

- Set: Collection of objects (distinct)

Note that this is an informal definition. If interested in some formalism, and why it is needed, look into axiomatic set theory (would be a whole new course).

- Those objects inside a set can be sets themselves (sets of sets)

- Fun fact: natural numbers can be interpreted as sets

Functions

- Formally, a function f from a set X to a set Y **is the set** of ordered pairs (x, y) such that x is in X, y is in Y and every element in x is the first component for exactly one ordered pair.

- Informally, a function is a process, and what we mentioned above is called the *graph* of the function

- X above is called the domain

- A sequence (or string) is a function with domain $\mathbb{N}$

Finite and Infinite Sets

- Informally, a set is finite if you can count it and finish

- Formally, S is finite if there exists a natural number n, and an injective function $f: S \to \{0,1, \dots, n\}$

- A set is infinite if it is not finite.

Or equivalently: S is infinite if there is an injective function $g: \mathbb{N} \to S$

Cardinality

- Two sets are said to have the same cardinality (equinumerous) if there is a bijection between them

- The cardinality of a set A is denoted by $|A|$

- A Cardinality is actually an equivalence class of the relation of equinumerosity

Comparing Cardinalities

- $|A| \leq |B|$ if there is an injection (injective function) from *A* to *B.*
- Such injection could be a bijection, in which case we have $|A| = |B|$
- For every set *S*, the set $P(S)$ (of all subsets of *S*) has a strictly larger cardinality than *S.* I.e. $|S| < |P(S)|$ (there is no bijection)

Some sets are more infinite than others

- $|\mathbb{N}| < |P(\mathbb{N})| < \left|P\big(P(\mathbb{N})\big)\right| < \cdots$

- A set *A* is countable if $|A| \leq |\mathbb{N}|$ (so it can be finite or infinite)

- A set *A* is uncountable if it is not countable.

In other words, if there is an injection of the natural numbers into *A*, but no injection of *A* into the natural numbers.

- Clearly uncountable is always infinite

The Continuum Hypothesis (just for fun)

- Can you find a set $|A|$ such that $|\mathbb{N}| < |A| < |P(\mathbb{N})|$?

Ans: Yes and No

- Don’t mix computable, countable, uncountable, not computable
- Uncomputable = non-computable = not computable
- In the realm of computability, WLOG, we will only be dealing with countable sets (subsets of $\mathbb{N}$)
- Recall, recursive functions and Turing machines deal with objects which can be coded as natural numbers

Back to where we finished last lecture

- We saw how we can give programs numbers (e.g. via Gödel numbering)
- We let P_e denote the e^{th} Turing program, and φ_e the corresponding partial computable function (in one variable)
- This implies that the set of all Turing Machines is countably infinite (infinite and countable)

The Universal Turing Machine

- There exists a TM U which if given input (e, x) it runs the eth TM with input x.

- Follows from CT

Infinite Computable

- A set is infinite computable if it is infinite and also computable
- Red V neck T-shirt: A T-shirt which is red and has a V neck

Infinite Computable is Diophantine

- Indeed, Diophantine = C.E.
- Computable >> C.E.
- Thus, Computable >> Diophantine
- Infinite & Computable >> Diophantine

The empty set is computable

- It is finite and every finite set is computable (why?)

- Or more directly: the characteristic function of the empty set is the zero function which is in computable (even more, it is initial in PRIM)

Prove that: If A is computable, then it is c.e. (decidable >> listable)

Proof1:

I_A is computable (given).

Recall: a set is c.e. if it is empty or is the range of a computable function.

If A is empty, then it is c.e. (implication holds by definition).

Assume $A \neq \emptyset$. We want to find a computable function f such that $range(f) = A$.

Since A is non-empty, there must be some $a \in A$. Fix such an a.

Let f be the function defined as follows

$$f(x) = \begin{cases} x & if\ I_A(x) = 1 \\ a & if\ I_A(x) = 0 \end{cases}$$

Proof2:

We describe a program that enumerates A which by CT can be mimicked by a Turing machine.

```
i  = 0
c = 0
While i==0:
        if I_A(c) = 1: #this runs a sub-program
                print(c)
                c = c+1
```

C.E. but not Computable (FINALLY)

Let $K = \{x: \varphi_x(x) \downarrow\}$

- Show that K is c.e. (Think)

- Show that K is NOT computable

- Assume towards a contradiction that *K* is computable.
- Consider the following function:

$$f(x) = \begin{cases} undefined & if\ x \in K \\ 0 & o.w \end{cases}$$

This *f* is partial computable because it can be mimicked by a TM:

1. we can computably decide if *x* is in *K* or not.
2. If *x* is in *K*, go in an infinite loop
3. If *x* is not in *K*, output 0

- But then, f must have a Gödel number, say e. I.e. $f = \varphi_e$

- If $e \in K$, then

$\varphi_e(e) = f(e) \uparrow$ i.e. $e \notin K$ (contradiction)

- If $e \notin K$, then

$\varphi_e(e) = f(e) = 0$ i.e. $\varphi_e(e) \downarrow$ i.e. $e \in K$ (contradiction)

What can you say about $\overline{K}$?

Remarks

- There are uncountably many non-computable subsets of $\mathbb{N}$
- This is because there are only countably many computable sets (why?)
- The same applies to the bigger class of c.e. sets. There are only countably many such sets.
- This means that the class of c.e. sets is very small

More about c.e. sets

- We defined a set to be c.e. if it is empty or the range of a computable function
- One can also show it is the range of a partial computable function (exercise)
- One can also show it is the **domain** of a partial computable function
- All are equivalent definitions

Proof:

Let A be a c.e. set

If A is empty, then A is the domain of the empty function given by the program which doesn't halt on any input

If A is not empty, then it is the range of a computable function, say $A = \{f(0), f(1), f(2), \dots\}$.

Let $\varphi(x) = \mu y[f(y) = x]$. Then $dom(\varphi) = A$

Computable Relations

- Recall, a binary relation over sets *X*, *Y* is a subset of the Cartesian product
$$X \times Y$$
- More generally, an *n*-ary relation over sets $X_1, \ldots, X_n$ is a subset of
$$X_1 \times \cdots \times X_n$$
- An *n*-ary relation on $\mathbb{N}$ is one for which $X_1 = \cdots = X_n = \mathbb{N}$
- A relation on $\mathbb{N}$ is computable if it is computable as a set
- We say a relation is c.e. if it is c.e. as a set.

Example

- $R = \{(x, y, z) \in \mathbb{N}^3 \colon x < y \ and \ z = 2x\}$

We have $R(1,2,2), R(0,3,0), R(10,11,20)$
But $\neg R(0,2,2), \neg R(0,0,0), \neg R(10,11,11)$

Here $\neg$ means negation

- R is clearly computable. There's a program which when given any tuple (a, b, c) it can decide if $R(a, b, c)$ or $\neg R(a, b, c)$

- Note that we can regard relations as Boolean valued functions

- $R_2 = \{(x, e) \in \mathbb{N}^2\colon\ \varphi_e(x) \downarrow\}$

Not computable (why?)

But it is c.e. because if $R_2(x, e)$ then you can confirm that computably

Special Cases

- Note that a function is a binary relation
- A non-empty subset of X is a unary (1-ary) relation on X.
- There are 0-ary relations (TRUE and FALSE)
- There is the empty relation $\emptyset$ which is the same as FALSE (holds for nothing)

Deeper analysis of $\varphi_e(x)\downarrow$

- Recall that $\varphi_e(a)\downarrow$ means that the partial computable function φ_e is defined at *a*, or equivalently, that the program P_e halts when given *a* as an input

- Consider now the following new notation $\varphi_{e,s}(x)\downarrow$. It means the computation halts within *s* steps (or stages)

- $\varphi_e(x)\downarrow$ iff $\exists s\ \varphi_{e,s}(x)\downarrow$

- Note that, for any fixed *s* the relation $\{(e,x){:}\,\varphi_{e,s}(x)\downarrow\}$ is computable unlike $\{(e,x){:}\,\varphi_e(x)\downarrow\}$ as we mentioned before

- Actually, the following ternary relation is computable

$$\{(e,s,x){:}\,\varphi_{e,s}(x)\downarrow\}$$

- In general, one can prove that:

A relation $R(x,y)$ is c.e. iff there exists a computable relation $C(a,x,y)$ such that for all *x,y*

$$R(x,y) \iff \exists a\, C(a,x,y)$$

The Arithmetical Hierarchy

- We use Σ_1^0 to denote the class of relations (formulas) obtained as $\exists \bar{a}\ C(\bar{a}, \bar{x})$ using some computable relation C

- Π_1^0 denotes the class of relations (formulas) obtained as $\forall \bar{a}\ C(\bar{a}, \bar{x})$ using some computable relation C

- Note that if a set is Σ_1^0 then its complement is Π_1^0 , and vice versa

Going higher

- Π_2^0 denotes the class of relations (formulas) obtained as $\forall \bar{a} \exists \bar{b}\ C(\bar{a}, \bar{b}, \bar{x}\ )$ using some computable relation C

Or equivalently $\forall \bar{a}\ D(\bar{a}, \bar{x}\ )$ for some Σ_1^0 relation D

- Σ_2^0 denotes the class of relations (formulas) obtained as $\exists \bar{a} \forall \bar{b}\ C(\bar{a}, \bar{b}, \bar{x}\ )$ using some computable relation C

In general

- Π^0_{n+1} denotes the class of relations (formulas) obtained as $\forall \bar{a}\ D(\bar{a}, \bar{x}\ )$ for some Σ^0_n relation *D*

- Σ^0_{n+1} denotes the class of relations (formulas) obtained as $\exists \bar{a}\ D(\bar{a}, \bar{x}\ )$ for some Π^0_n relation *D*

- Note that, for all *n*, $\Sigma^0_n \cup \Pi^0_n \subsetneq \Sigma^0_{n+1} \cap \Pi^0_{n+1}$

- Recall we mentioned that

A relation $R(x, y)$ is c.e. iff there exists a computable relation $C(a, x, y)$ such that for all *x,y*

$$R(x, y) \iff \exists a\, C(a, x, y)$$

- This means that C.E. = Σ_1^0

- BTW, Computable = $\Sigma_0^0 = \Pi_0^0$

The Normal Form Theorem for C.E. Sets

- The following are equivalent:

- A is c.e.
- A is Σ_1^0
- A = W_e for some $e \in \mathbb{N}$

Relative Computability

- We have just seen that C.E. = Σ_1^0
- How about Σ_2^0 ? Or more generally, Σ_{n+1}^0?
- Are they c.e. in some sense w.r.t. some higher level?
- Indeed, it is all about the computable function which enumerates the set

Oracle Machines and Relative Computability

- A set A is Σ_2^0 means that it is either empty or the range of a Σ_1^0 function f

- More clearly, f can be computed with a program which has access to, e.g., the set K we described earlier

- Such program is given the knowledge of the indicator function of K