Computability Theory

Seen problems that are hard (eg EXPTIME)
but some are <u>impossible</u>.

Key example: the Halting Problem

$\text{HALT}(f, x)$:= will the function f
halt on input x
(i.e., no infinite loop)

<u>First:</u> HALT is NP-hard.

Want $3SAT \leq_p \text{HALT}$.

Given instance x of $3SAT$
Let f be a (slow) algorithm for $3SAT$.

```
def f'(x):
    if f(x) == False:
        while 1: pass
    else:
        return
```

Then: $\text{HALT}(f', x) = \text{True} \Leftrightarrow f(x) = \text{True} \Leftrightarrow$
x a true instance.
f' takes $O(1)$ bits to describe $\Rightarrow 3SAT \leq_p \text{HALT}$
$\Rightarrow$ HALT is NP-hard.

But we didn't use anything about 3-SAT other than that it is computable.

For <u>any</u> computable problem q,
$q \preceq_p$ HALT by same argument

[=> Halt is $\underbrace{\text{EEEE}}_{2^{2^{2^{2^n}}} \text{ time}}$xPTime-hard]

So HALT is maximally hard; but actually, it is impossible:

```
def g(x):
    if HALT(x, x):
        while 1: true
    else: return
```

If HALT can be implemented, the program g() has some fixed size.

Q: does g(g) ever halt?

Paradox: YES => NO; NO => YES.

=> Cannot compute HALT in finite time.

Busy Beaver Numbers

What about the following way to solve HALT:

- Run program for $L = 10^{10^{10}}$ steps
- if not terminated, output False

Correct if L is large enough: longer than time to compute $f(x)$.

If (f, x) takes n bits to describe, there are 2^n inputs of length n.

time until termination:

$$T_1, \ldots, T_{2^n}$$ ← some ∞ rest finite

Let $$B(n) := \max_{T_i < \infty} T_i$$

```
def Halt(f, x):
    L = B(|(f,x)|)
    Run f(x) for L steps
    output True if f(x) terminated.
```

This solves the Halting problem
The halting problem is not computable

$\Rightarrow$ $B(n)$ is not computable.
[nor any upper bound on it.]

Unprovability

Of course, the halting problem is solvable for any finite input size:

Hard code $B(1000000) \leq \overbrace{L = 999\ldots9}^{\text{ridiculously long}}$
to solve on inputs $\leq$ 1000000 bits long

But... how can you know if you've written enough 9's?
A: Provably impossible to tell.

Gödel's 2nd incompleteness Theorem:
No consistent set of axioms can prove their own consistency

["Consistent" means cannot prove contradiction]

Modern math is built on ZFC axioms.
$\Rightarrow$ impossible to prove ZFC consistent.
[we hope! if it is possible, modern math is broken.]

```
def FindContradiction():
    For all s ∈ {0,1}*:
        check if s is valid ZFC proof of contradiction
        if so, return.
```

Halt(FindContradiction) = False $\Leftrightarrow$ ZFC consistent
$\Rightarrow$ impossible to prove FindContradiction doesn't halt
$\Rightarrow$ $B(|\text{FindContradiction}|)$ cannot be proven.
Q: how big is |FindContradiction|?

$S(n)$ = max # steps any n-state Turing machine takes on blank binary tape before halting

$S(n)$ behaves like $B(n)$, but more formally specified
[so we can evaluate specific n]

$S(2) = 6$, $S(3) = 21$, $S(4) = 107$

$S(5) = 47$ million, probably.
[assuming ~ 15 turing machines run forever]

$S(6) \gg 10^{36000}$
$S(7) \gg 10^{10^{10^{10^{10^6}}}}$
Surely vast underestimates

⋮

$S(n)$ uncomputable in general

$S(|\text{FindContradiction}|)$ computable (its a fixed constant)
but not provably correct.

Scott Aaronson & undergrad: $|\text{FindContradiction}| < 8000$
more recent: < 2000
$S(2000)$: no upper bound can be proven in principle
$S(741)$: upper bound proves (hard part of) Riemann Hypothesis

Models of Computation

Turing machines
Church Numerals / lambda calculus

Theorem:

Functions computable by Turing
= Functions computable by Church

Church-Turing Thesis: (conjecture)
= functions computable by <u>any</u> "physical process"
or "human following algorithm"

Extended Church-Turing Thesis:

P is the same in all processes

[Probably false: Quantum computers (BQP) $\neq$ P ?]