Computability Theory

Seen problems that are hard (e.g. EXPTIME) but some are impossible.

Key example: the Halting Problem

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

**First:** HALT is NP-hard.

Want \( 3\text{SAT} \leq_p \text{HALT}. \)

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

\[
\text{def } f'(x): \\
\text{if } f(x) = \text{False:} \\
\text{while } 1: \text{ Pass} \\
\text{else:} \\
\text{return}
\]

**Then:** \( \text{HALT}(f', x) = \text{True} \iff f(x) = \text{True} \)

\( f' \) takes \( O(1) \) bits to describe \( \Rightarrow 3\text{SAT} \leq_p \text{HALT} \)

\( \Rightarrow \text{HALT} \) is NP-hard.
But we didn’t use anything about 3-SAT other than that it is computable.

For any computable problem \( g \) \( \exists p \) HALT by same argument

\[
[\neg \text{Halt } i) \quad \text{EEEEE EPT}\text{ime - hard]}
\]

\[
\text{time}
\]

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

```python
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: \( \text{YES} \rightarrow \text{NO} \quad \text{NO} \rightarrow \text{YES} \).

\( \Rightarrow \) cannot compute HALT in finite time.
Busy Beaver Numbers

What about the following way to solve \textsc{Halt}: 

- Run program for \(L = n^{10^{10}}\) steps
- If not terminated, output \text{False}

Correct if \(L\) is large enough longer than time to compute \(F(k)\).

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

Time until termination:

\[
T_1, \quad \ldots, \quad T_{2^n} \in \text{rest finite}
\]

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

```python
def \textsc{Halt}(f, x):
    L = B(\lceil \log_2 f(x) \rceil)
    Run \(f(x)\) for \(L\) steps
    Output \text{True} if \(f(x)\) terminated.
```

This solves the Halting problem
The halting problem is not computable

\[\Rightarrow B(n) \text{ is not computable.}\]

[Nor any upper bound on it.]
Of course, the halting problem is solvable for any finite input size:

Hard code $B(1000000) \leq L = 9999999$ 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.

$\implies$ impossible to prove ZFC consistent.

[we hope! if it is possible, modern math is broken.]

```
def Find_Contradiction():
    For all $s \in \mathbb{N}$:
        Check if $s$ is valid ZFC proof or contradiction.
        if so, return.
```

$\text{Halt( Find_Contradiction )} = \text{False} \iff \text{ZFC consistent}$

$\implies$ impossible to prove Find_Contradiction doesn't halt

$\implies$ $B(\text{Find_Contradiction})$ cannot be proven.

Q: how big is $|\text{Find_Contradiction}|$?
\[ S(n) = \max \# \text{ steps any } n\text{-state Turing machine takes on blank binary tape before halting} \]

\[ S(n) \text{ behaves like } B(n), \text{ but more formally specified} \]

\[ S(2) = 6, \quad S(3) = 21, \quad S(4) = 107 \]

\[ S(5) = 4.7 \text{ million, probably.} \]

\[ \text{[Assuming } \sim 15 \text{ turing machines run forever]} \]

\[ S(6) \gg 10^{36,000} \quad \text{< Surely vast underestimates} \]

\[ S(7) \gg 10^{10^{10^{10^{10^6}}}} \]

\[ S(n) \text{ uncomputable in general} \]

\[ S(1 \text{ Find Contradiction}) \text{ computable (it's a fixed constant) but not provably correct.} \]

Scott Aaronson undergar! Find Contradiction! \(< 8000 \)

more recent : \(< 2000 \)

\[ S(2000): \text{ no upper bound can be proven in principle} \]

\[ S(741): \text{ upper bound proves (had first) Riemann Hypothesis} \]
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 any "physical process"
= Functions computable by any "human following algorithm"

Extended Church-Turing Thesis:
P is the same in all processes

[Probably false: quantum computers (BQP) ≠ P ?]