CS 121: Lecture 15
More Uncomputability

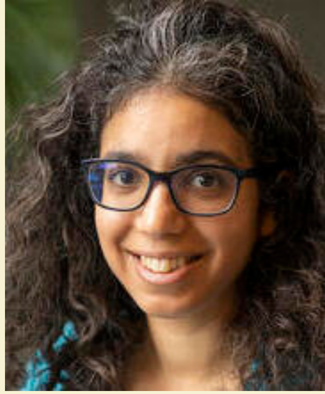
Madhu Sudan

https://madhu.seas.Harvard.edu/courses/Fall2020

Book: https://introtcs.org

How to contact us
- The whole staff (faster response): CS 121 Piazza
- Only the course heads (slower): cs121.fall2020.course.heads@gmail.com

Announcements:



- Advanced Section: Nada Amin: Uncomputability & PL Design
- Thanks for feedback.
  - Confirm – are breakouts no good?
  - TFs scouring the feedback also!
- Sections: Week 7 cycle start, material on canvas (as usual).

Where we are:

Part I: Circuits:
Finite computation,
quantitative study

Part II: Automata:
Infinite restricted computation,
quantitative study

Part III: Turing Machines:
Infinite computation, qualitative study

Part IV: Efficient Computation:
Infinite computation, quantitative study

Part V: Randomized computation:
Extending studies to non-classical algorithms

Review of last lecture

- # of functions = uncountable $= |\mathbb{R}|$
- # of computable functions = countable. $= |\{0,1\}^*| = |\mathbb{N}|$
- So … ∃ an uncomputable function
- Further $\mathrm{Cantor}(M) = \overline{M(M)}$ uncomputable

$f : \{0,1\}^* \to \{0,1\}$

This lecture (& next)

- Uncomputability much more pervasive
- “Intent of a program” uncomputable

Today: HALT is uncomputable

- Definition: HALT(M,x) = 1 if M halts on input x; 0 otherwise.

$$\text{Halt}: \underset{\substack{\uparrow \\ M}}{\{0,1\}^*} \times \underset{\substack{\uparrow \\ x}}{\{0,1\}^*} \to \{0,1\}$$

- 2 Proofs:
  - Diagonalization
  - Reduction from CANTOR

$\exists$ 1-1 $\quad E : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$

use "prefix-free encodings"

Proof 1 (Direct Diagonalization):

- Let A be a TM that solves HALT, i.e., $\forall M, x,\ \ A(M,x) = \text{HALT}(M,x)$
- Consider the following Algorithm (which has equivalent TM – HOCAEIT)

$B(z)$:
Compute $A(z,z)$
If $A(z,z) = 1$ then loop forever
Else Halt and output 1.

- Note: We are defining B but not running it! It does not have to halt (in fact crucial that it does not on some inputs.
- Key point: B is a TM.

Proof 1 (Direct Diagonalization):

- Let A be a TM that solves HALT, i.e., $\forall M, x,\ \ A(M,x) = \mathrm{HALT}(M,x)$
- Consider B

$B(z)$:
Compute $A(z,z)$
If $A(z,z) = 1$ then loop forever
Else halt and output 1.

- What is $A(B,B)$?
  - Case 1: $A(B,B) = 1 \Rightarrow$ (by correctness of A) B halts on input B
    $\Rightarrow$ (by construction of B) B loops forever $\Rightarrow$ Contradiction.

B(B) ? B(B) ↗

Proof 1 (Direct Diagonalization):

- Let A be a TM that solves HALT, i.e., $\forall M, x,\ \ A(M, x) = \mathrm{HALT}(M, x)$
- Consider B

$B(z)$:
Compute $A(z, z)$
If $A(z, z) = 1$ then loop forever
Else halt and output 1.

- What is $A(B, B)$?
  - Case 1: $A(B, B) = 1 \Rightarrow$ (by correctness of A) B halts on input B

    $B(B)$

    $\Rightarrow$ (by construction of B) B loops forever $\Rightarrow$ Contradiction.
  - Case 2: $A(B, B) = 0 \Rightarrow$ (by correctness of A) B does not halt on input B

    $\Rightarrow$ (by construction of B) B halts on B (outputs 1) $\Rightarrow$ Contradiction!

    $B(B)$

Thoughts:

- Very slick!
- But just an implementation of Diagonalization. (Note $B(B)$; $A(z,z)$ ...)
- Food for thought: What happens if A does not always halt but correctly determines $\mathrm{HALT}(M,x)$ on inputs where it halts?

Godel's Incompleteness

Either PL is not capable of executing every computable function or, it can not be checked for HALT.

Proof 2: (General) Reduction

"clearly only way to compute G is by computing F"

- Reductions: Key theme in Computer Science
  - Function F reduces to G ($F \leq G$) if algorithm for G implies algorithm for F
  - How to prove it?

```
Alg-F(x):
Blah Blah Blah
 z =  Alg-G(y)
Blah blah blah
```

    - Build algorithm for F using Alg-G as subroutine.
    - Alg-F correctly computes F if Alg-G correctly computes G

DO NOT USE

- Usual Interpretation: Positive:
  - Somebody builds tools for mean, median; I just invoke it on my data with wrapper.
- Our Use: Negative:
  - Start with F known not to have algorithm. Infer G does not!
    - Do you remember any so far in this course?

Example: HALT uncomputable

$G = \text{HALT}$
$F = \text{CANTOR}$

- Recall CANTOR uncomputable.
- Will use this to prove HALT uncomputable.
- So what do we need to do?

Alg-$F(x)$:
Blah Blah Blah
$z =$ Alg-$G(y)$
Blah blah blah

Halt is uncompt

CANTOR is computable $\Leftarrow$ Halt is computable

CANTOR is uncomputable

Example: $\mathrm{\mathbf{HALT}}$ uncomputable

- Recall CANTOR uncomputable.
- Will use this to prove HALT uncomputable.
- So what do we need to do?

Alg-CANTOR(x):
Blah Blah Blah
$z =$ Alg-HALT(y)
Blah blah blah

ALG-CANTOR

- Recall $\mathrm{CANTOR}(M) = \overline{M(M)}$

Alg-CANTOR(M):
  $b \leftarrow$ Alg-HALT(M, M)
  If $b = 0$ output 1
  Else run M on M and let output be c
    Output $\bar{c}$

- Claim 1: Alg-CANTOR always halts if Alg-HALT correct.
- Claim 2: Alg-CANTOR correctly computes CANTOR.

- Claim 1+Claim 2: Alg-CANTOR computes (the uncomputable function) CANTOR if Alg-HALT exists $\Rightarrow$ Alg-HALT does not exist $\Leftrightarrow$ HALT uncomputable.

What did we prove?

- CANTOR $\leq$ HALT ? Or HALT $\leq$ CANTOR?

Computable < CANTOR & (CANTOR $\leq$ HALT)

Computable < Halt

(Basic) Reduction

- For many problems we will use a very basic reduction (even simpler than $\mathrm{CANTOR} \leq \ \mathrm{HALT}$)

Alg-F(x):
$y = R(x)$
$Return$ Alg-G(y)

Example:

- $E(M) = 1 \Leftrightarrow$ ~~$\forall x, \quad M(x) = 0$ or M does not halt on x~~

  $\exists z, \quad M(z) = 1$

- $\text{HALT} \leq E$

Alg-HALT(M, x):
Define M_x as follows:

> $M_x(z)$: Ignore z,
> output 1 if M halts on x
> output 0 o.w.

$Return$ Alg-E(M_x)

Section + Next Lecture

- More Uncomputability + Reductions
  - HALT-ON-ZERO
    - H-O-Z(M) = 1 if M accepts “” and 0 otherwise.
    - Moral: It is not the infinity of inputs that makes HALT hard!
  - Rice’s theorem
    - Every non-trivial semantic property of algorithms is uncomputable!