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
- # of computable functions = countable.
- So ... ∃ an uncomputable function
- Further Cantor(M) = \( \overline{M(M)} \) uncomputable
This lecture (& next)

• Uncomputability much more pervasive
• “Intent of a program” uncomputable
Today: HALT is uncomputable

- Definition: $\text{HALT}(M,x) = 1$ if $M$ halts on input $x$; 0 otherwise.

- 2 Proofs:
  - Diagonalization
  - Reduction from CANTOR
Proof 1 (Direct Diagonalization):

• Let $A$ be a TM that solves $HALT$, i.e., $\forall M, x, A(M, x) = HALT(M, x)$

• 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 $\text{HALT}$, i.e., $\forall M, x, A(M, x) = \text{HALT}(M, x)$
- 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.
Proof 1 (Direct Diagonalization):

- Let $A$ be a TM that solves \textsc{Halt}, i.e., $\forall M, x, A(M, x) = \text{Halt}(M, x)$

- 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.
  - 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!
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 $\text{HALT}(M, x)$ on inputs where it halts?
Proof 2: (General) Reduction

• 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
  \[ z = \text{Alg-}G(y) \]
  Blah blah blah

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

• 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

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

\[
\text{Alg-}F(x): \\
\text{Blah Blah Blah} \\
z = \text{Alg-}G(y) \\
\text{Blah blah blah}
\]
Example: **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
```
Recall $\text{CANTOR}(M) = \overline{M(M)}$

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.

Alg-CANTOR($M$):
\[
\begin{align*}
& b \leftarrow \text{Alg-HALT}(M, M) \\
& \text{If } b = 0 \text{ output 1} \\
& \text{Else run } M \text{ on } M \text{ and let output be } c \\
& \text{Output } \overline{c}
\end{align*}
\]
What did we prove?

• CANTOR \leq HALT ? Or HALT \leq CANTOR?
(Basic) Reduction

• For many problems we will use a very basic reduction (even simpler than CANTOR \leq HALT)

Alg-F(x):
\[ y = R(x) \]
Return Alg-G(y)
Example:

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

- \( \text{HALT} \leq E \)

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

\[ M_x(z): \text{Ignore } z, \]
\[ \text{output 1 if } M \text{ halts on } x \]
\[ \text{output 0 o.w.} \]

Return Alg-E\((M_x)\)
• 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!