CS 121: Lecture 16
Rice’s Theorem

Adam Hesterberg

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

Book: https://intrototcs.org

How to contact us

The whole staff (faster response): CS 121 Piazza

Only the course heads (slower): cs121.fall2020.course.heads@gmail.com
If eligible, 
Get Ready to Vote

If you’re voting by mail, make sure to submit your absentee ballot or take action if your ballot has not arrived. Learn more at bit.ly/harvardvotebymail.

If you’re voting in person, make sure to pick a date/time, research your polling location and voter ID laws. Learn more at bit.ly/harvardvoteinperson.

Research what’s on your ballot: ballotpedia.org.

If you have voting questions or need help, text “@votinghelp” to 81010 or email voteschallenge@harvard.edu
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 two lectures

- \( \text{Cantor}(M) = \overline{M(M)} \) uncomputable
- \( \text{HALT}(M, x) \) uncomputable
- \( E(M) = 1 \Leftrightarrow \forall x, \ M(x) = 0 \) or \( M \) does not halt on \( x \): uncomputable
This lecture

- Uncomputability much more pervasive
  - We’ll keep doing examples until most of the class votes “Go Faster”.
- “Intent of a program” uncomputable
Thm 1: $COMPUTESXOR(M) = 1$ iff $M(x) = XOR(x)$ for all $x$. Then $COMPUTESXOR$ is uncomputable.

Thm 2: $ONLYODD(M) = 1$ iff $|M(x)|$ odd for all $x$. Then $ONLYODD$ is uncomputable.

Thm 3: $NOEVENS(M) = 1$ unless $|M(x)|$ even for some $x$. Then $NOEVENS$ is uncomputable.

Thm 4: $HALTONSHORT(M) = 1$ iff $M(x)$ halts whenever $|x| \leq 100$. Then $HALTONSHORT$ is uncomputable.

Thm 5: $MONOTONE(M) = 1$ iff $|M(x) \leq M(x')|$ whenever $x \preceq x'$. Then $MONOTONE$ is uncomputable.

Thm 6: $COMPUTESPAL(M) = 1$ iff $M(x) = PAL(x)$ for all $x$. Then $COMPUTESPAL$ is uncomputable.
Thm 1: \( COMPUTESXOR(M) = 1 \) iff \( M(x) = XOR(x) \) for all \( x \). Then \( COMPUTEXOR \) is uncomputable.

- Recall: \( HALT \) is uncomputable.
- Will use this to prove \( COMPUTESXOR \) is uncomputable.
- I.e. if we could solve \( COMPUTESXOR \), we could solve \( HALT \).
- I.e. we’ll reduce from \( HALT \) to \( COMPUTEXOR \).
- I.e. we’ll rule out the possibility (\( HALT \) hard, \( COMPUTEXOR \) easy)
- I.e. we’ll show that \( HALT \leq COMPUTEXOR \).

Alg-HALT(\( x \)):  
Blah Blah Blah  
\( z = \text{Alg-COMPUTEXOR}(y) \)  
Blah blah blah
"COMPUTES\text{ }XOR\text{ }uncomputable" \text{ doesn't say:}

- ...doesn't say that there's no machine that computes XOR.
- i.e. doesn't say that XOR is uncomputable.
- E.g. Alg-XOR:

\begin{algorithm}
\textbf{Alg-XOR}(x):
\text{ ans } = 0 \\
\text{ for bit in } x: \\
\hspace{1cm} \text{ ans } = \text{ ans } \text{ XOR bit} \\
\text{ return ans}
\end{algorithm}
Proof of Thm 1 (COMPUTESXOR uncomp.)

- **HALT ≤ COMPUTEXOR**
- Suppose there exists an algorithm $ALG$ – COMPUTEXOR...

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

\[ M_x(y) : \text{Simulate } M \text{ on } x. \]
\[ \text{Ignore the result.} \]
\[ \text{Return } ALG\text{-XOR}(y). \]

\[ z = \text{Alg-COMPUTEXOR}(M_x) \]
\[ \text{Return } z \]

- This would be an algorithm computing HALT, which doesn’t exist!
Break: discuss proof: COMPUTESXOR uncomp.

- HALT ≤ COMPUTEXOR
- Suppose there exists an algorithm ALG – COMPUTEXOR...

\[
\text{Alg-HALT}(M, x):
\text{Define } M_x \text{ as follows:}

M_x(y): \text{Simulate } M \text{ on } x.
\text{Ignore the result.}
\text{Return ALG-XOR(y)}.
\]

\[
z = \text{Alg-COMPUTESXOR}(M_x)
\text{Return } z
\]

- This would be an algorithm computing HALT, which doesn’t exist!
Thm 2: $\text{ONLYODD}(M) = 1$ iff $|M(x)|$ odd for all $x$. Then $\text{ONLYODD}$ is uncomputable.

- Recall: $\text{HALT}$ is uncomputable.
- Will use this to prove $\text{ONLYODD}$ is uncomputable.
- I.e. if we could solve $\text{ONLYODD}$, we could solve $\text{HALT}$.
- I.e. we’ll reduce from $\text{HALT}$ to $\text{ONLYODD}$.
- I.e. we’ll rule out the possibility ($\text{HALT}$ hard, $\text{ONLYODD}$ easy)
- I.e. we’ll show that $\text{HALT} \leq \text{ONLYODD}$.

Alg-$\text{HALT}(x)$:
Blah Blah Blah
\[ z = \text{Alg-ONLYODD}(y) \]
Blah blah blah
"**ONLYODD** uncomputable" doesn’t say:

- ...doesn’t say that there’s no machine that outputs odd-length strings on every input.
- i.e. doesn’t say that if a function has only odd-length output, it’s uncomputable.
- E.g. Alg-ODD:

```plaintext
Alg-ODD(x):
  ignore x.
  return 1001111
```

Proof of Thm 2 (ONLYODD uncomp.)

- $\text{HALT} \leq \text{ONLYODD}$
- Suppose there exists an algorithm $\text{ALG} - \text{ONLYODD}$...

**Alg-HALT($M_x$):**
Define $M_x$ as follows:

- $M_x(y)$: Simulate $M$ on $x$.
- Ignore the result.
- Return $\text{Alg-ODD}(y)$.

$z = \text{Alg-ONLYODD}(M_x)$
Return $z$

Goal: make a machine $M_x$ such that $M$ halts on $x$ iff $M_x$ computes an odd-output function.

- This would be an algorithm computing HALT, which doesn’t exist!
Thm 3: \( NOEVEN(M) = 1 \) unless \( |M(x)| \) even for some \( x \). Then \( NOEVEN \) is uncomputable.

- Recall: \( HALT \) is uncomputable.
- Will use this to prove \( NOEVEN \) is uncomputable.
- I.e. if we could solve \( NOEVEN \), we could solve \( HALT \).
- I.e. we’ll reduce from \( HALT \) to \( NOEVEN \).
- I.e. we’ll rule out the possibility (\( HALT \) hard, \( NOEVEN \) easy)
- I.e. we’ll show that \( HALT \leq NOEVEN \).

\[
\text{Alg-HALT}(x):
\begin{align*}
\text{Blah Blah Blah} \\
&
z = \text{Alg-NOEVEN}(y) \\
&\text{Blah blah blah}
\end{align*}
\]
“NOEVENS uncomputable” doesn’t say:

- ...doesn’t say that there’s no machine that outputs only odd-length strings or doesn’t halt.
- i.e. doesn’t say that if a function has no even-length output, it’s uncomputable.
- E.g. Alg-LOOP:

```
Alg-LOOP(x):
while(True)
```

- ...doesn’t say the opposite, either!
- i.e. doesn’t say that if a function has even-length output, it’s uncomputable.
- E.g. Alg-EVEN:

```
Alg-EVEN(x):
return ```121_is_great```
```
Proof of Thm 3 (NOEVENS uncomp.)

- $\text{HALT} \leq \text{NOEVENS}$
- Suppose there exists an algorithm $\text{ALG} – \text{NOEVENS}$...
Thm 4: \( \text{HALTONSHORT}(M) = 1 \) iff \( M(x) \) halts whenever \( |x| \leq 100 \). Then \( \text{HALTONSHORT} \) is uncomputable.

- Recall: \( \text{HALT} \) is uncomputable.
- Will use this to prove \( \text{HALTONSHORT} \) is uncomputable.
- I.e. if we could solve \( \text{HALTONSHORT} \), we could solve \( \text{HALT} \).
- I.e. we’ll reduce from \( \text{HALT} \) to \( \text{HALTONSHORT} \).
- I.e. we’ll rule out the possibility (\( \text{HALT} \) hard, \( \text{HALTONSHORT} \) easy)
- I.e. we’ll show that \( \text{HALT} \leq \text{HALTONSHORT} \).

\[
\text{Alg-HALT}(x):
\text{Blah Blah Blah}
\]
\[
z = \text{Alg-HALTONSHORT}(y)
\]
\[
\text{Blah blah blah}
\]
“HALTONSHORT uncomputable” doesn’t say:

- ...doesn’t say that there’s no machine that halts on short inputs.
- E.g. Alg-EVEN:

```python
Alg-EVEN(x):
return "121_is_great"
```

- ...doesn’t say the opposite, either!
- E.g. Alg-LOOP:

```python
Alg-LOOP(x):
while(True)
```
Proof of Thm 4 (HALTONSHORT uncomp.)

- **HALT \leq HALTONSHORT**
- Suppose there exists an algorithm *ALG* — *HALTONSHORT*...

```
Alg-HALT(M, x):
Define \( M_x \) as follows:
\[
\begin{align*}
M_x(y) & : \text{Simulate } M \text{ on } x. \\
& \text{Ignore the result.} \\
& \text{Return } \text{Alg-EVEN}(y).
\end{align*}
\]
\[
\begin{align*}
z = \text{Alg-HALTONSHORT}(M_x) \\
\text{Return } z
\end{align*}
```

- This would be an algorithm computing HALT, which doesn’t exist!

\( \text{Goal: } \text{HALTONSHORT}(M_x) \) is different depending on whether \( M \) halts on \( x \).

Halts on short inputs if \( M \) halts on \( x \).
Rice’s Theorem

**Rice’s Thm:** For every \( F: \{0,1\}^* \rightarrow \{0,1\} \), if \( F \) is semantic then either \( F = \text{one} \) or \( F = \text{zero} \) or \( F \) is uncomputable.

**Def:** \( M \) and \( M' \) are functionally equivalent if for every \( x \in \{0,1\}^* \), \( M(x) = M'(x) \)
Notation: \( M \cong M' \)

**Def:** \( F: \{0,1\}^* \rightarrow \{0,1\} \) is semantic if for every \( M \cong M', F(M) = F(M') \)

**Q:** Let \( \text{one}: \{0,1\}^* \rightarrow \{0,1\} \) be constant one function (\( \text{one}(w) = 1 \) for every \( w \)). Then \( \text{one} \) is semantic.
**Rice’s Theorem:** If $F : \{\text{Turing Machines}\} \rightarrow \{0, 1\}$ has property that $F(M)$ is *semantic* (only depends on what $M$ computes, not how) and $F$ is not *trivial* (true for every $M$ or no $M$), then $F$ is uncomputable.
Recall: $HALT$ is uncomputable.
Will use this to prove $F$ is uncomputable.
I.e. if we could solve $F$, we could solve $HALT$.
I.e. we’ll reduce from $HALT$ to $F$.
I.e. we’ll rule out the possibility ($HALT$ hard, $F$ easy)
I.e. we’ll show that $HALT \leq F$.

Alg-$HALT(x)$:
Blah Blah Blah
$z = \text{Alg-}F(y)$
Blah blah blah
“\( F \) isn’t trivial” means:

- There’s some machine \( M \) such that \( F(M) = 1 \).
- E.g. Alg-???:

\[ \text{ Alg-at-least-it-exists}(x): \]

??? has property ???

\[ \uparrow \]

A machine that computes XOR

(These might be switched.)

- There’s some machine \( M \) such that \( F(M) = 0 \).
- E.g. Alg-LOOP:

\[ \text{ Alg-LOOP}(x): \]

while(True)

\[ \uparrow \]

A machine that halts on short input
Proof of Thm n (Rice’s theorem: F uncompl.)

- $\text{HALT} \leq F$
- Suppose there exists an algorithm $ALG - F$...

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

$M_x(y)$: Simulate $M$ on $x$.
Ignore the result.
Return $\text{Alg-at-least-it-exists}(y)$.

$z = \text{Alg-F}(M_x)$
Return $z$

- This would be an algorithm computing HALT, which doesn’t exist!
Break: discuss Rice’s Theorem & proof

- \( \text{HALT} \leq F \)
- Suppose there exists an algorithm \( ALG = F \)...

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

\[
M_x(y): \text{Simulate } M \text{ on } x. \\
\text{Ignore the result.} \\
\text{Return Alg-at-least-it-exists}(y).
\]

\[z = \text{Alg-F}(M_x)\]

Return \( z \)

- This would be an algorithm computing HALT, which doesn’t exist!
Rice’s Theorem caveats

**Rice’s Thm:** For every $F: \{0,1\}^* \to \{0,1\}$, if $F$ is semantic then either $F = \text{one}$ or $F = \text{zero}$ or $F$ is uncomputable.

A first approximation is “functions that take a TM $M$ as input are uncomputable”, but:

- Check whether you actually needed the TM as input.

- Functions that ask about *how* $M$ computes things might or might not be computable. “$M$ has <100 states” vs “$M$ has <100 states and computes HALT”

...this isn’t just a loophole; it lets us salvage things we want from uncomputability!
Things we’d like to do, but can’t.

**Bug checking:** Given a program M (say, in Python), check whether it will always return.

**Type checking:** Given a program M (say, in Python), check whether it ever calls the `concat` function with inputs that aren’t strings.

**Equivalence checking:** A clever programmer claims to have found a faster replacement for your function. It’s fast, but you don’t understand the code. Is it right?

Given T.M. M & input x, m ∈ without using concat, simulate M on x.

Suppose we could with Alg. A.
Rice’s Theorem (Fundamental Theorem of Software Verification):

Every semantic $F$ is either trivial or uncomputable.

Is software verification doomed?

Q: Let $\text{ValidType}: \{0,1\}^* \rightarrow \{0,1\}$ be function that maps a C program $P$ to 1 iff when $P$ is executed, it will never call a function with a char parameter with an int parameter.
Prove that $\text{ValidType}$ is computable.

Type mismatch error
Coping with Rice

- Turing Machines
- General programming languages
- \( \lambda \) calculus
- ...

<table>
<thead>
<tr>
<th>Proof systems</th>
<th>Type systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simply typed ( \lambda ) calculus</td>
<td></td>
</tr>
<tr>
<td>System F</td>
<td></td>
</tr>
<tr>
<td>HTML</td>
<td></td>
</tr>
<tr>
<td>Context Free Grammars</td>
<td></td>
</tr>
<tr>
<td>Regular expressions</td>
<td></td>
</tr>
<tr>
<td>ANSI SQL (92)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Turing Complete Models
+ Stronger
- No semantic analysis

Restricted Computational Models
- Weaker
+ Semantic analysis
Section + Next Lecture

- **Section: 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!

- **Next Lecture: Efficient Computation (P)**