CS 121: Lecture 16
Rice's Theorem

Adam Hesterberg

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

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.

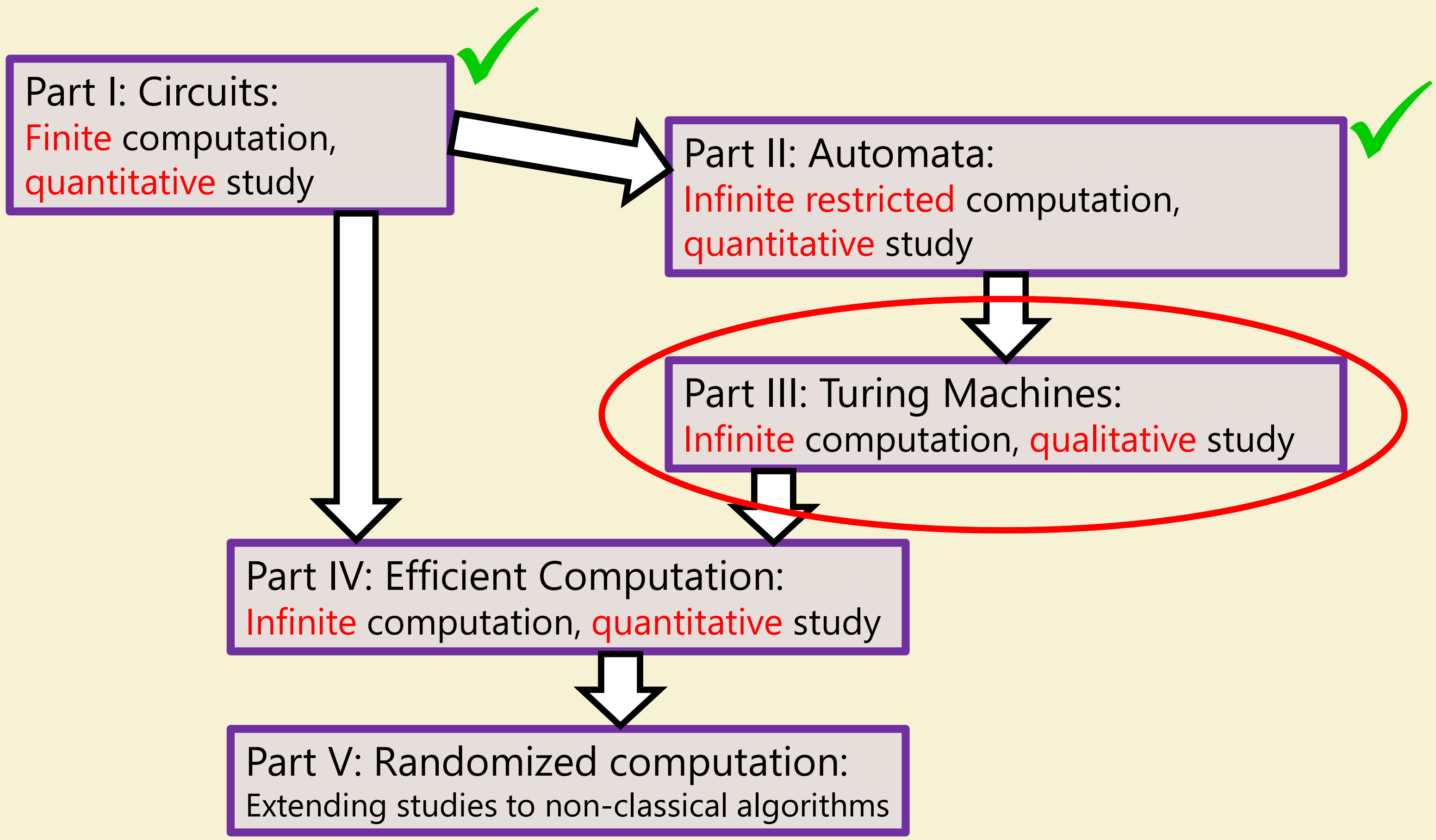
If you're **voting in person**, make sure to pick a date/time, research your polling location and voter ID laws. Learn more at bit.ly/harvardvoteinperson.

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**

HARVARD VOTES CHALLENGE

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
- HALT(M,x) uncomputable
- $E(M) = 1 \Leftrightarrow \ \forall x, \quad M(x) = 0$ 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 $COMPUTEXOR$ is uncomputable.

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

Thm 3: $NOEVENS(M) = 1$ <u>unlessss</u> $|M(x)|$ 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 $x \preccurlyeq x'$. Then $MONOTONE$ is uncomputable.

Thm 6: $\mathrm{COMPUTE}SPAL(M) = 1$ iff $M(x) = PAL(x)$ for all x. Then $COMPUTEPAL$ 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
$z =$ Alg-$COMPUTEXOR(y)$
Blah blah blah

"$COMPUTESXOR$ uncomputable" 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:

```
Alg-XOR(x):
ans = 0
for bit in x:
        ans = ans XOR bit
return ans
```

Proof of Thm 1 (COMPUTESXOR uncomp.)

- $\text{HALT} \leq COMPUTEXOR$
- Suppose there exists an algorithm $ALG - COMPUTEXOR$…

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

$M_x(y)$: Simulate M on x.
Ignore the result.
Return ALG-XOR(y).

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

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

Break: discuss proof: COMPUTESXOR uncomp.

- $\mathrm{HALT} \leq COMPUTEXOR$
- Suppose there exists an algorithm $ALG - COMPUTEXOR$…

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

> $M_x(y)$: Simulate M on x.
> Ignore the result.
> Return ALG-XOR(y).

$z =$ Alg-$\mathrm{COMPUTE}XOR(M_x)$
$Return\ z$

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

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

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

Alg-HALT(x):
Blah Blah Blah
$z =$ 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:

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

Proof of Thm 2 (ONLYODD uncomp.)

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

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

> $M_x(y)$: Simulate M on x.
> Ignore the result.
> Return Alg-ODD(y).

$z =$ Alg-ONLYODD(M_x)
Return z

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

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

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

```
Alg-HALT(x):
Blah Blah Blah
 z = Alg-NOEVENS(y)
Blah blah blah
```

"$NOEVENS$ 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.)

- $\text{HALT} \le NOEVENS$
- Suppose there exists an algorithm $ALG - NOEVENS$...

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

> $M_x(y)$: Simulate M on x.
> Ignore the result.
> Return 1-(Alg-EVEN(y)).

$z = $ Alg-NOEVENS(M_x)
Return z

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

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

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

```
Alg-HALT(x):
Blah Blah Blah
 z = Alg-HALTONSHORT(y)
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:

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

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

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

Proof of Thm 4 (HALTONSHORT uncomp.)

- $\text{HALT} \leq HALTONSHORT$
- Suppose there exists an algorithm $ALG - HALTONSHORT$…

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

$M_x(y)$: Simulate M on x.
Ignore the result.
Return Alg-EVEN(y).

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

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

Rice's Theorem

Rice's Thm: For every $F: \{0,1\}^* \to \{0,1\}$, if F is semantic then either $F = one$ or $F = zero$ 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: $F: \{0,1\}^* \to \{0,1\}$ is semantic if for every $M \cong M'$, $F(M) = F(M')$

Q: Let $one: \{0,1\}^* \to \{0,1\}$ be constant one function ($one(w) = 1$ for every w).
Then one is semantic.

Thm 3: $COMPUTEXOR(M) = 1$ iff $M(x) = XOR(x)$ for all x. Then $COMPUTEXOR$ is uncomputable.

Thm 4: $COMPUTEPAL(M) = 1$ iff $M(x) = PAL(x)$ for all x. Then $COMPUTEPAL$ is uncomputable.

Rice's Theorem: If $F\colon \{Turing\ Machines\} \to \{0,1\}$ 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.

Thm 7: $HALTONSHORT(M) = 1$ iff $M(x) \neq \perp$ whenever $|x| \leq 100$. Then $HALTONSHORT$ is uncomputable.

Thm n (Rice's Theorem)

- 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
$z =$ Alg-$F(y)$
Blah blah blah

"F isn't trivial" means:

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

```
Alg-at-least-it-exists(x):
???
```

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

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

(These might be switched.)

Proof of Thm n (Rice's theorem: F uncomp.)

- $\text{HALT} \leq F$
- 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 Alg-at-least-it-exists(y).

$z =$ Alg-F(M_x)
Return z

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

Break: discuss Rice's Theorem & proof

- $\mathrm{HALT} \leq F$
- 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 Alg-at-least-it-exists(y).

$z =$ 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 $F = one$ or $F = zero$ 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?

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 $ValidType$ is computable.

`Type mismatch error`

Coping with Rice

- Turing Machines
- General programming languages
- λ calculus
- …

Type systems

Proof systems

Simply typed λ calculus

System F

HTML

Context Free Grammars

Regular expressions

…

ANSI SQL (92)

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)