Complexity Theory

Instructor: Roei Tell

Computability

Course outline

Modeling

Computability

Basic complexity

P vs NP

Strong forms of P ≠ NP

Lemons into lemonade

Universal machine

Uncomputable problems

Reductions

Kolmogorov complexity

Computability theory

- Precursor to complexity theory, computers
- Inspired by logic & set theory in the early 20th century
  (Hilbert 10th problem, Gödel's incompleteness theorem)
- Disregards resources and asks "what can be computed?"

TMs as strings

- Claim:

  We can represent every TM as a binary string.

- Proof idea:
  - identify ∑ with [|∑|] and S with [|S|]
  - represent δ as a sequence of input-output values

TMs as strings

- From now on, fix some way of encoding TMs as binary strings
- Notation: <M> = encoding of M

Universal machine

- Definition:

  A TM U is a Universal Turing Machine if U(<M>, x) = M(x) .

  - intuition: it's a computer that runs your code
  - depends on how we encode TMs as binary strings

Universal machine

- Theorem:

  There is a universal TM U such that for every <M>,x , if M(x) halts after T steps then U(<M>, x) halts after poly(T) steps.

Universal machine

- Theorem:

  There is a universal TM U such that for every M, x, if M(x) halts after T(n) steps then U(<M>, x) halts after poly(T(n)) steps.

- From now on: fix an encoding + universal TM

Can computers do everything?

- Definition:

  We say that $f:\{0,1\}^* \to \{0,1\}^*$ is **uncomputable** if there's no TM that computes f .

  We say that $L \subseteq \{0,1\}^*$ is **undecidable** if there's no TM that decides L.

- Strongest possible lower bound!

Can computers do everything?

- Plan:

1. Uncomputable problems exist.
2. Explicit uncomputable problems.
3. Explicit and natural uncomputable problems.

Can computers do everything?

- Plan:

1. Uncomputable problems exist.
2. Explicit uncomputable problems.
3. Explicit and natural unc

There are natural problems that can't be algorithmically solved, no matter the resources.

Undecidable languages exist

- Theorem (informal):

  There are “more” languages than TMs.

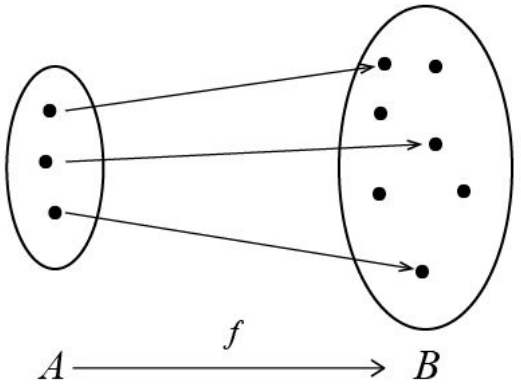
Undecidable languages exist

› Theorem (informal):

There are “more” languages than TMs.

› Corollary:

There is an undecidable language.



Undecidable languages exist

- Lemma:

  The set of all TMs is countable.

- Proof:

  There's an injective function from the set of TMs to $\{0,1\}^*$.

Undecidable languages exist

- Lemma:

  The set of all languages is uncountable.

- Proof:

  The set of all languages is the power set of $\{0,1\}^*$.

  By Cantor's diagonalization argument, it's uncountable.

Undecidable languages exist

- Corollary:

  There exists an undecidable language.

- Proof:

Explicit undecidable languages

- What happens when we give M its own description <M> as input?

Explicit undecidable languages

› What happens when we give M its own description <M> as input?

› Definition:

We say that M is **self-accepting** if M(<M>) = 1, and **self-rejecting** if M(<M>) = 0.

Explicit undecidable languages

- Question:

  Can we determine if a machine is self-rejecting?

- Part of a more general challenge:

  Look at a machine's description, determine something meaningful about its **functionality.**

Explicit undecidable languages

› Theorem:

The language DIAG = { <M> : M is self-rejecting } is undecidable.

› Proof:

[1] when <M> isn't a valid encoding of a TM, then <M> ∉ DIAG

Explicit undecidable languages

- Observation:

  A TM may never reach s_{end} on some inputs x, in which case M(x) is undefined.

- Examples:
  - infinite loop (we've all been there)
  - silly machine: "keep moving right forever"

Explicit undecidable languages

- Extending the definitions:
  - When M doesn't halt on input x, we say that M(x) is undefined
  - When there's an n-bit input x such that M(x) is undefined, we say that $T(n) = \infty$

Explicit undecidable languages

- Theorem:

  The language HALT = { <M> : M(<M>) halts } is undecidable.

- Proof:

Explicit undecidable languages

› So far we capitalized on the liar's paradox ("sarvam mithyā bravīmi")

"I am lying right now" ⇐ is this true or false?

S = " ¬S " ⇐ is S a true statement?

› Computational versions of Gödel's incompleteness theorem

Explicit natural undecidable languages

› Definition:

The universal language is U = { <M>,x : M(x) = 1 } .

[1] when <M> isn't a valid encoding of a TM, then (<M>,x) ∉ U

Explicit natural undecidable languages

- Definition:

  The universal language is U = { <M>,x : M(x) = 1 } .

- Note (confusing naming clash):

  A universal machine isn't guaranteed to decide the universal language. It's only guaranteed that if M(x) stops after T steps then U(<M>,x) stops after poly(T) steps, but if M(x) doesn't stop, U(<M>,x) may not stop!

Explicit natural undecidable languages

- Theorem:

  The universal language is undecidable.

- Proof:

Explicit natural undecidable languages

› Definition:

For a machine M, denote by $L(M) = \{ x : M(x) = 1 \}$.

› Note (edge case):

If M(x) is undefined (since M doesn't halt on x) then $x \notin L(M)$.

Explicit natural undecidable languages

- We will consider collections $\mathbb{\Pi}$ of languages (i.e., $\mathbb{\Pi} \subseteq 2^{\{0,1\}^*}$)
- Examples:

  $\mathbb{\Pi}$ = { all languages L such that $0 \in L$ }

  $\mathbb{\Pi}$ = { all languages L that contain all prime numbers }

  $\mathbb{\Pi}$ = { all languages L that don't contain your first name }

- Given M, can we decide if $L(M) \in \mathbb{\Pi}$?

  ("does M accept 0?", "does M accept prime numbers?", ...)

Explicit natural undecidable languages

› Rice's Theorem:

For any non-trivial collection $\mathbb{\Pi}$ of languages,

$$L_{\mathbb{\Pi}} = \{ \langle M \rangle : L(M) \in \mathbb{\Pi} \}$$

is undecidable.

[1] non-trivial: there is a M st $L(M) \in \mathbb{\Pi}$, there is M' st $L(M') \notin \mathbb{\Pi}$

Explicit natural undecidable languages

› Rice's Theorem:

For any non-trivial collection $\mathbb{\Pi}$ of languages,

$$L_{\mathbb{\Pi}} = \{ \langle M \rangle : L(M) \in \mathbb{\Pi} \}$$

is undecidable.

We can't algorithmically understand <u>anything meaningful</u> about the functionality of a given TM!

[1] non-trivial: there is a M st $L(M) \in \mathbb{\Pi}$, there is M' st $L(M') \notin \mathbb{\Pi}$

Explicit natural undecidable languages

- Modern intuition:
  - can TMs be in an obfuscated form, hiding their functionality?
  - lemons into lemonade: if this is true, we can hide what our code is doing from adversaries!

- Carrying this intuition through requires more advanced complexity and cryptography theory.

Kolmogorov complexity

› Question:

Which of the following strings has more **information content?**

1100010101000010011001000110100000001010

0100110001110000111100000111110000000111111

0101010101010101010101010101010101010101

0000000000000000000000000000000000

Kolmogorov complexity

› Definition:

The **Kolmogorov complexity** of a string x is

$$K(x) = \min_{\text{TM } M,\ \text{binary string } w}\{\ |\langle M\rangle, w| : M(w) = x\ \}$$

[1] the K-complexity is sometimes defined using a universal TM or with M but without w.

Kolmogorov complexity

› Definition:

The **Kolmogorov complexity** of a string x is

$$K(x) = \min_{\text{TM } M,\ \text{binary string } w}\{\ |\langle M\rangle, w| : M(w) = x\ \}$$

VERY DIFFERENT from the rest of the course: a TM for a single string, rather than for a function!

[1] the K-complexity is sometimes defined using a universal TM or with M but without w.

Kolmogorov complexity

› Definition:

The **Kolmogorov complexity** of a string x is

$$K(x) = \min_{\text{TM } M,\ \text{binary string } w}\{\ |\langle M\rangle, w| : M(w) = x\ \}$$

› for K-complexity, we encode <M>,w such that after the delimiter "," we use one bit to represent each bit of w (i.e., without using tricks to simulate a larger alphabet)

[1] the K-complexity is sometimes defined using a universal TM or with M but without w.

Kolmogorov complexity

- Theorem:

  For every $x \in \{0,1\}^*$, $K(x) \leq |x| + O(1)$

- Theorem:

  For every integer n, most n-bit strings x have $K(x) \geq n - 1$.

[1] the constant in the “O(1)” notation depends on our encoding of TMs.

Kolmogorov complexity

- Theorem:

  The function K(x) is uncomputable.

Kolmogorov complexity

- Theorem:

  The function K(x) is uncomputable.

- Idea:

  Berry paradox:

  “The smallest positive integer not definable in under a million letters”

Kolmogorov complexity

- Theorem:

  The function K(x) is uncomputable.

- Proof:

Verifiable problems

- Recall:

  L is **decidable** (“computable”) if there’s a TM that decides L.

- We proved:
  - HALT is undecidable
  - U is undecidable // U is the universal problem
  - K is uncomputable // K is Kolmogorov complexity
  - …

Verifiable problems

› Definition:

We say that $L \subseteq \{0,1\}^*$ is **verifiable** if there's a TM V st for all $x \in \{0,1\}^*$

$x \in L \quad \Rightarrow \quad \exists\ w \quad V(x,w) = 1$

$x \notin L \quad \Rightarrow \quad \forall\ w \quad V(x,w) = 0$

$\forall\ w \qquad V(x,w)$ halts

› The definition extends to funcs $f:\{0,1\}^* \to \{0,1\}^*$ in the obvious way

Verifiable problems

- Theorem:
  - HALT is verifiable
  - U is verifiable // U is the universal problem
- Proof:

Verifiable problems

- Takeaways:
  - notion of a proof system (two-player game)
  - it can be easier to verify a solution than to find one!

Verifiable problems

- Recall (from Rice's theorem):

  $L(M) = \{ x : M(x) = 1 \}$ $\Leftarrow$ "the set of inputs M accepts"

  If M doesn't halt on some x (or outputs gibberish), then $x \notin L(M)$.

- Theorem:

  L is verifiable $\Leftrightarrow$ there is a TM M such that $L(M) = L$.

Verifiable problems

- For comparison:
  - L decidable ⇔ there is a TM deciding the language L.
  - L verifiable ⇔ there is a TM M such that L(M) = L.

Verifiable problems

- Definition:

  Let $\mathcal{R}$ = { L ⊆ {0,1}* : L is decidable } // "recursive"

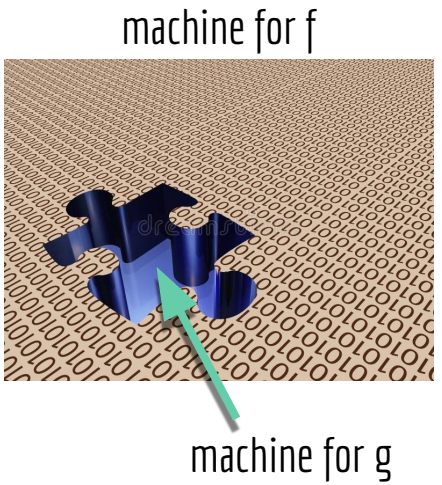
  Let $\mathcal{RE}$ = { L ⊆ {0,1}* : L is verifiable } // "recursively enumerable"

- We proved:
  - there are problems in $\mathcal{RE} \setminus \mathcal{R}$ (e.g., HALT, U)
  - L ∈ $\mathcal{RE}$ ⇔ ∃ TM M st L(M) = L

Reductions

- Recap:
  - if we have a machine for g, we can build a machine for f

    " f reduces to g ", " f ≤ g "



Reductions

- Recap:
  - if we have a machine for g, we can build a machine for f

    " f reduces to g ", " f ≤ g "
  - g easy ⇒ f easy
  - f hard ⇒ g hard