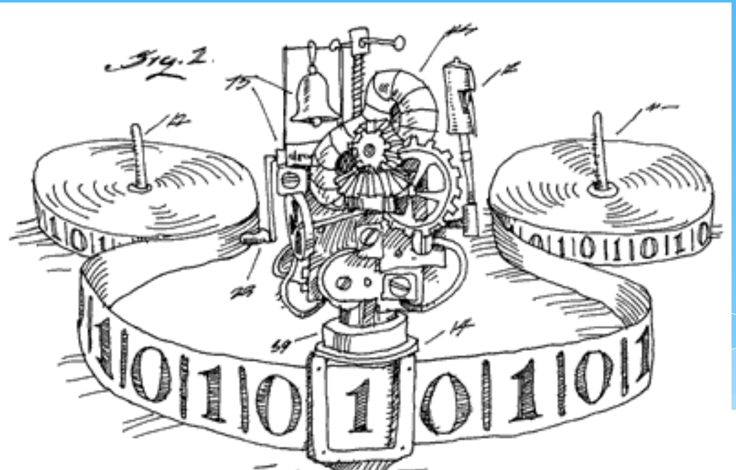
EECS 376: Foundations of Computer Science

Chris Peikert

13 February 2023



Agenda

- Recognizability and Unrecognizability
- Proving unrecognizability
- Data compression and Kolmogorov complexity

Quote of The Day

"Recognizing isn't at all like seeing; the two often don't even agree."

– Sten Nadolny

EECS ECE CSE

Decidability and Recognizability

- **Definition (Recall):** A program/TM M ***decides*** a language A if:
  - $x \in A \implies M$ accepts x
  - $x \notin A \implies M$ rejects x
  - M is called a **decider**.
- **Consequence:** A decider halts on any input.
- **Definition (Recall):** L is ***decidable*** if some TM decides L.
- **Definition (Recall):** The ***language*** of M, $L(M) = \{x : M \text{ accepts } x\}$
- **Definition:** M ***recognizes*** a language A if $A = L(M)$. That is:
  - $x \in A \implies M$ accepts x
  - $x \notin A \implies M$ does not accept x (i.e., rejects or loops)
  - M is called a ***recognizer***.
- **Observation:** L_{ACC} is _undecidable_, but _recognizable_. (Why?)

Why is Recognizability Useful?

- A "one-sided" notion: recognizer for L must accept any $x \in L$, and not accept any $x \notin L$, but need not answer in the latter case.
- $L_{\text{GOOD-IDEA}} = \{x : x \text{ is ASCII for a good idea}\}$
  - x="Chris should go to Cedar Point this summer"
  - x="Chris should learn to skateboard"
  - x="Chris should be a contestant on Survivor"
    - Recognizer: let me think about it…

EECS ECE CSE

Properties of Recognizable Languages

- **Definition:** A language A is ***recognizable*** if there exists a program M (a "recognizer") that recognizes it: L(M)=A.
- **Warm-up**: Let A_1, A_2 be *recognizable* languages. Must $B = A_1 \cup A_2 = \{x : x \in A_1 \text{ or } x \in A_2\}$ be recognizable?
- **Answer:** Some programs M_1 and M_2 recognize A_1 and A_2, resp'ly. We describe a program M for B.
- $M(x)$:
  - Run M_1 on x (as a "subroutine")
  - Run M_2 on x (as a "subroutine")
  - *Accept* if at least one of the above accepts, else *reject*
- **Problem:** This does not work! (Why?)
- **(Recall)** listing $\mathbb{Z}$ **Attempt 1:** $0,1,2,3, \ldots, -1, -2, -3, \ldots$
  Does not work.
- **Attempt 2:** *Alternate* between positive and negative:
  $0,1,-1,2,-2,\ldots$

Properties of Recognizable Languages

- $M(x)$:
  - Alternate between executions of M_1 on x and M_2 on x, one step at a time
  - *Accept* if (and as soon as) either of the programs accepts
  - *Reject* if both programs reject
- **Analysis:**
  - If $x \in A_1 \cup A_2$ then one of the programs accepts x, so M accepts x
  - If $x \notin A_1 \cup A_2$ then neither program ever accepts x, so M does not accept x
    - Both programs reject $x \implies M$ rejects x
    - At least one of the programs loops on $x \implies M$ loops on x
- **Conclusion:** The class of recognizable languages is closed under union (and intersection—why?).

Properties of Recognizable Languages

- **Q:** Is the class of recognizable languages closed under complement?
- **In other words:** Let A be recognizable. Is $\bar{A} = \Sigma^* \backslash A$ recognizable?
- **Theorem:** If a language A and its complement $\bar{A}$ are both *recognizable*, then A is *decidable*.
- **Proof:** Suppose programs M_1 and M_2 *recognize* A and $\bar{A}$, respectively.
- Since M_1 recognizes A:
  - $x \in A \implies M_1$ accepts x
  - $x \notin A \implies M_1$ rejects or loops on x
- Since M_2 recognizes $\bar{A}$:
  - $x \in A \implies M_2$ rejects or loops on x
  - $x \notin A \implies M_2$ accepts x

Properties of Recognizable Languages

- **Q:** Is the class of recognizable languages closed under complement?
- **In other words:** Let A be recognizable. Is $\bar{A} = \Sigma^* \backslash A$ recognizable?
- **Theorem:** If a language A and its complement $\bar{A}$ are both *recognizable*, then A is *decidable*.
- **Proof:** Suppose programs M_1 and M_2 *recognize* A and $\bar{A}$, respectively. We describe a program that *decides* A. $M(x)$:
  - Alternative between executions of $M_1(x)$ and $M_2(x)$
  - If $M_1(x)$ accepts, accept.
  - If $M_2(x)$ accepts, reject.
- **Analysis:**
  - $x \in A \implies M_1$ accepts $x \implies M$ accepts x
  - $x \notin A \implies x \in \bar{A} \implies M_2$ accepts $x \implies M$ rejects x

EECS M ECE CSE

Proving Unrecognizability

* **Theorem:** If A and its complement $\bar{A}$ are both *recognizable*, then A is *decidable*.
* **Contrapositive:** If A is undecidable, then *at least one of* A or $\bar{A}$ is unrecognizable.
* **Corollary:** If A is undecidable but recognizable, then $\bar{A}$ is unrecognizable.
* **Theorem:** $\overline{L_{\mathsf{ACC}}} = \{\langle M, x\rangle : M$ does not accept $x\}$ is unrecognizable:
  * L_{ACC} is undecidable (last week)
  * L_{ACC} is recognizable *('interpreter' TM U recognizes L_{ACC})*
  * $\implies \overline{L_{\mathrm{ACC}}}$ is unrecognizable

Another Example

* **Theorem:** Let $L_{\text{GOOD}} = \{\langle M \rangle : \text{OSU} \notin L(M)\}$. L_{GOOD} is unrecognizable.
* **Proof:** We show that L_{GOOD} is undecidable via $L_{\text{ACC}} \leq_T L_{\text{GOOD}}$, and that L_{GOOD} is recognizable.
* **Step 1:** Let G be a membership oracle for L_{GOOD}. We construct a decider C for L_{ACC} as follows:
* $C(\langle M, x \rangle)$:
  1. Construct a new program $M'(w)$ = "ignore w, run $M(x)$ and answer as M does."
  2. Call $G(\langle M' \rangle)$ and output the *opposite* of G's output.
* **Analysis:** C halts since G does. Moreover:
  * M acc x $\iff$ M' acc OSU $\iff$ G rej $\langle M' \rangle$ $\iff$ C acc

Another Example

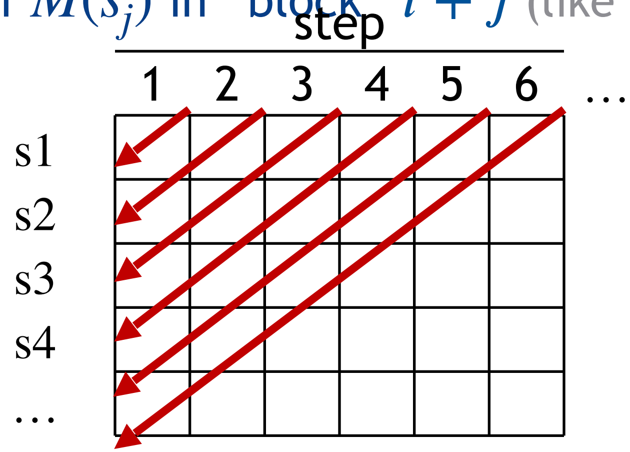
- **Theorem:** Let $L_{\text{GOOD}} = \{\langle M \rangle : \text{OSU} \notin L(M)\}$. L_{GOOD} is unrecognizable.
- **Proof:** We show that $\overline{L_{\text{GOOD}}}$ is undecidable via $L_{\text{ACC}} \leq_T \overline{L_{\text{GOOD}}}$, and that $\overline{L_{\text{GOOD}}}$ is recognizable.
- **Step 2:** Build recognizer R for $\overline{L_{\text{GOOD}}} = \{\langle M \rangle : \text{OSU} \in L(M)\}$:
  1. Run M on OSU and answer as M does.
- **Analysis:**
  - $\langle M \rangle \in \overline{L_{\text{GOOD}}} \Rightarrow M$ accepts OSU $\implies R$ accepts $\langle M \rangle$
  - $\langle M \rangle \notin \overline{L_{\text{GOOD}}} \Rightarrow M$ doesn't accept OSU $\implies R$ doesn't accept $\langle M \rangle$
- **Conclusion:** $\overline{L_{\text{GOOD}}}$ is recognizable but undecidable, so L_{GOOD} is unrecognizable.

$L_\emptyset$ is Unrecognizable

- **Claim:** $L_\emptyset = \{\langle M \rangle : L(M) = \emptyset\}$ is unrecognizable.
- **Proof:** We show that $L_\emptyset$ is undecidable ($L_{\text{ACC}} \leq_T L_\emptyset$) and $\overline{L_\emptyset}$ is recognizable.
- **Step 1:** Let N be an oracle for $L_\emptyset$. Construct a decider C for L_{ACC}:
- $C(\langle M, x\rangle)$:
  1. Construct a program "$M'(w)$: run M on x and answer as M does"
  2. Call N on $\langle M' \rangle$ and return the *opposite* output
- **Analysis:** C halts since N does. Moreover:
  - M acc $x \iff L(M') \neq \emptyset \iff N$ rej $\langle M' \rangle \iff C$ acc $(\langle M \rangle, x)$

$L_\emptyset$ is Unrecognizable

- **Claim:** $L_\emptyset = \{\langle M \rangle : L(M) = \emptyset\}$ is unrecognizable.
- **Proof:** We show that $L_\emptyset$ is undecidable ($L_{\text{ACC}} \leq_T L_\emptyset$) and $\overline{L_\emptyset}$ is recognizable.
- **Step 2:** We need to construct a recognizer for $\overline{L_\emptyset} = \{\langle M \rangle : L(M) \neq \emptyset\}$.
- **Observation:** We just need to find a single input that M accepts (if any). But there are (countably) infinitely many inputs $s_j \in \Sigma^*$, and M can loop!
- **Idea:** Do step i of $M(s_j)$ in "block" $i + j$ (like in proof that $\mathbb{Q}^+$ is countable).



"Dovetailing"

- **Claim:** $L_\emptyset = \{\langle M \rangle : L(M) = \emptyset\}$ is unrecognizable.
- **Proof:** We show that $L_\emptyset$ is undecidable ($L_{\mathrm{ACC}} \leq_T L_\emptyset$) and $\overline{L_\emptyset}$ is recognizable.
- **Step 2:** We need to construct a recognizer for $\overline{L_\emptyset} = \{\langle M \rangle : L(M) \neq \emptyset\}$.
- **Idea:** Do step i of $M(s_j)$ in "block" $i + j$ (like in proof that $\mathbb{Q}^+$ is countable).
- $R(\langle M \rangle)$:
  1. For $t = 1, 2, 3, \ldots$:
     2. For $j = 1, 2, \ldots, t$:
        3. Run one (additional) step of $M(s_j)$
        4. If $M(s_j)$ accepts, <u>accept</u>.
- **Analysis:**
  - $\langle M \rangle \in \overline{L_\emptyset} \Rightarrow \exists j, k .\ M$ accepts s_j in k steps $\Longrightarrow R$ accepts $\langle M \rangle$.
  - $\langle M \rangle \notin \overline{L_\emptyset} \Rightarrow L(M) = \emptyset \Longrightarrow R$ loops on $\langle M \rangle$.

Summary: Recognizability

- **Summary:** We can show that a language L is unrecognizable by showing that L is undecidable and that $\overline{L}$ is recognizable.
- **Definition:** L is ***co-recognizable*** if $\overline{L}$ is recognizable.
- **Observation 1:** Decidable languages are exactly those that are recognizable and co-recognizable. (Why?)
- **Observation 2:** There are countably infinitely many languages that are recognizable or co-recognizable. (Why?)
- **Conclusion:** "Almost all" languages are neither recognizable nor co-recognizable!
  - **Example:** L_{EQ}. (Prove this!)

Data Compression

- **Problem:** Encode some given data D using fewer bits.
- **Lossless:** compress so that D is recoverable exactly
- **Example:** GIF, PKZIP
- **Lossy:** some "small deviations" are acceptable
- **Example:** JPEG, MPEG

Kolmogorov Complexity

- Let w be a binary string and let U be FPL (e.g., TMs).
- **Definition:** The ***Kolmogorov Complexity*** of w:
  $$K_U(w) = \min\left\{ \left|\langle M\rangle\right| : M \text{ outputs } w, \text{ on empty input } \varepsilon \right\}.$$
  That is, the size of the shortest U-program that outputs w.
- **Claim 1:** For every U and every $w \in \{0,1\}^*$: $K_U(w) = O\big(|w|\big)$.
- **Examples:**
  - **C++:** "cout << w" : $|w|$ + |<iostream>| + 8*8
  - **Python:** "print(w)" : $|w|$ + 7*8
  - **TMs:** a state for every symbol of w : $|w|$ + |7-tuple|.

Thought Experiment

- **Definition:** Let 0^m denote the string consisting of m zeros.
- **Question 1:** Is "cout << $00\ldots0$" a shortest program to output 0^m?
- **Question 2:** What is the size |"cout << 0^m"| as a function of m?
- **Answer:** $\Theta(m)$
- **Definition:** For each m define a new program P_m:
  - int main() { for (int i=0; i<m; ++i) putchar('0'); }
- **Question:** What is the size $|P_m|$ as a function of m?
- **Answer:** $O(\log m)$: m is hardcoded in binary.

EECS ECE CSE

Kolmogorov Complexity

- **Claim 2:** For every U and n, $\exists w \in \{0,1\}^n$ s.t. $K_U(w) \geq n$.
- **Proof:** Counting argument (details left as an exercise)
- **Implication:** For every n, every lossless compression program has an "uncompressible" string (file) of size n.
- **Claim 3:** $K_U(w)$ is uncomputable (for any "expressive enough" U)
- **Definition:** A function f is ***computable*** if there exists a program that, given w as input, halts and outputs $f(w)$.
  - For a TM, outputting a value means writing it on its tape.
- **Reworded:** No program can compute the function $K_U(w)$ *(correctly on <u>every</u> binary string w)*

Berry's Paradox

* **Aside (Berry's Paradox):** "The first positive integer that cannot be defined in <70 characters."
* Let $S \subset \mathbb{Z}^+$ be the set of positive integers that cannot be defined in <70 characters. Let $x = \min(S)$. Then:
  * x cannot be defined in <70 characters, since $x \in S$
  * x can be defined by the sentence "The first positive integer that cannot be defined in <70 characters.", which has 68 characters

Paradox!

$K_U(w)$ is Uncomputable

- **Berry's Paradox:** "The first positive integer that cannot be defined in <70 characters."
- **Strategy:**
  - Assume $K_U(w)$ is computable by some program M.
  - **Recall:** There is a length-n string x s.t. $K_U(x) \geq n$.
  - Define a program Q_n that uses M to output the *first* string w_n of length n for which $K_U\left(w_n\right) \geq n$.
  - But $|Q_n| = O(\log n) < n$, and since Q_n outputs w_n, we have $K_U(w_n) \leq |Q_n| < n$.

Paradox!

$K_U(w)$ is Uncomputable

- **Proof:** Pick the language U. Suppose M is a program that computes $K_U(w)$.
- **Definition:** For every n define new program Q_n:
  - const int LENGTH = n; *(n is "hardcoded")*
  - Iterate over all $x \in \{0,1\}^{\mathrm{LENGTH}}$:
    - Compute $K_U(x)$ *(using M as a black-box)*
    - Output the first x such that $K_U(x) \geq$ LENGTH

n encoded in binary

- **Observation:** For every n the size of the program Q_n is $O(\log n)$.
- **Analysis:** Let w_n be the output of Q_n. What is $K_U(w_n)$?
- **By definition:** $K_U(w_n) \geq n$, since Q_n outputs an x such that $K_U(x) \geq n$.

  On the other hand, Q_n outputs w_n, so Q_n is a program that outputs w_n; by definition of Kolmogorov complexity, $K_U(w_n) \leq |Q_n|$.
- **Therefore:** $K_U(w_n) \geq n$ and $K_U(w_n) \leq \mathrm{O}(\log n)$
- **Contradiction:** Conditions cannot be fulfilled simultaneously!
- **Conclusion:** No such M exists.

Applications

* ***Full Employment Theorem for Compiler Writers:*** There does not exist a perfect size-optimizing compiler.
* **Alternate Version:** There is no perfect program analysis.
* **Why:**
  1. Will produce the shortest program
  2. Can detect infinite loops and replace them with one line
* **Conclusion:** Many fundamental problems in CS are undecidable!
* **Industry Implications:** Intel, Google, IBM have entire divisions dedicated to tackling these problems in important special cases. But there are no general perfect solutions!