BU CS 332 – Theory of Computation

https://forms.gle/miNx7n2WQrvEZXYf9

Lecture 14:

- Undecidability
- Reductions

Reading:

Sipser Ch 4.2, 5.1

Mark Bun

March 20, 2024

Where we are and where we’re going

Church-Turing thesis: TMs capture all algorithms

Consequence: studying the limits of TMs reveals the limits of computation

Last time: Countability, uncountability, and diagonalization

Today: Existential proof that there are undecidable and unrecognizable languages

An explicit undecidable language

Reductions: Relate decidability / undecidability of different problems

Last time: A general theorem about set sizes

Theorem: Let X be any set. Then the power set $P(X)$ does **not** have the same size as X.

Proof: Assume for the sake of contradiction that there is a surjection $f: X \rightarrow P(X)$

Goal: Construct a set $S \in P(X)$ that cannot be the output $f(x)$ for any $x \in X$

Diagonalization argument

Assume a surjection $f: X \rightarrow P(X)$

x	$x_1 \in f(x)$?	$x_2 \in f(x)$?	$x_3 \in f(x)$?	$x_4 \in f(x)$?	…
x_1	Y	N	Y	Y	
x_2	N	N	Y	Y	
x_3	Y	Y	Y	N	
x_4	N	N	Y	N	
$\vdots$					$\ddots$

Define S by flipping the diagonal:

Put $\quad x_i \in S \quad \Longleftrightarrow \quad x_i \notin f(x_i)$

Last time: A general theorem about set sizes

Theorem: Let X be any set. Then the power set $P(X)$ does **not** have the same size as X.

Proof: Assume for the sake of contradiction that there is a surjection $f: X \to P(X)$

Construct a set $S \in P(X)$ that cannot be the output $f(x)$ for any $x \in X$:

$$S = \{x \in X \mid x \notin f(x)\}$$

If $S = f(y)$ for some $y \in X$,

then $y \in S$ if and only if $y \notin S$

Undecidable Languages

Undecidability / Unrecognizability

Definition: A language L is **undecidable** if there is no TM deciding L

Definition: A language L is **unrecognizable** if there is no TM recognizing L

An existential proof

Theorem: There exists an undecidable language over $\{0, 1\}$

Proof:

Set of all encodings of TM deciders: $X \subseteq \{0, 1\}^*$

Set of all languages over $\{0, 1\}$:

a) $\{0, 1\}$
b) $\{0, 1\}^*$
c) $P(\{0, 1\}^*)$: The set of all subsets of $\{0, 1\}^*$
d) $P(P(\{0, 1\}^*))$: The set of all subsets of the set of all subsets of $\{0, 1\}^*$

An existential proof

Theorem: There exists an undecidable language over $\{0, 1\}$

Proof:

Set of all encodings of TM deciders: $X \subseteq \{0, 1\}^*$

Set of all languages over $\{0, 1\}$: $P(\{0, 1\}^*)$

There are more languages than there are TM deciders!

$\Rightarrow$ There must be an undecidable language

An existential proof

Theorem: There exists an unrecognizable language over $\{0,1\}$

Proof:

Set of all encodings of TMs: $X \subseteq \{0,1\}^*$

Set of all languages over $\{0,1\}$: $P(\{0,1\}^*)$

There are more languages than there are TM recognizers!

$\Rightarrow$ There must be an unrecognizable language

“Almost all” languages are undecidable

But how do we actually find one?

An Explicit Undecidable Language

Our power set size proof

Theorem: Let X be any set. Then the power set $P(X)$ does **not** have the same size as X.

1) Assume, for the sake of contradiction, that there is a surjection $f: X \to P(X)$

2) "Flip the diagonal" to construct a set $S \in P(X)$ such that $f(x) \neq S$ for every $x \in X$

3) Conclude that f is not onto, a contradiction

Specializing the proof

Theorem: Let X be the set of all TM deciders. Then there exists an undecidable language in $P(\{0,1\}^*)$

1) Assume, for the sake of contradiction, that $L: X \to P(\{0,1\}^*)$ is a surjection

2) "Flip the diagonal" to construct a language $UD \in P(\{0,1\}^*)$ such that $L(M) \neq UD$ for every $M \in X$

3) Conclude that L is not onto, a contradiction

An explicit undecidable language

TM M					
M_1					
M_2					
M_3					
M_4					
$\vdots$					

Why is it possible to enumerate all TMs like this?

a) The set of all TMs is finite
b) The set of all TMs is countably infinite
c) The set of all TMs is uncountable

An explicit undecidable language

TM M	$M(\langle M_1 \rangle)$?	$M(\langle M_2 \rangle)$?	$M(\langle M_3 \rangle)$?	$M(\langle M_4 \rangle)$?		$D(\langle D \rangle)$?
M_1	Y	N	Y	Y	…	
M_2	N	N	Y	Y		
M_3	Y	Y	Y	N		
M_4	N	N	Y	N		
$\vdots$					$\ddots$	
D						

$UD = \{\langle M \rangle \mid M \text{ is a TM that does not accept on input } \langle M \rangle\}$

Claim: UD is undecidable

An explicit undecidable language

Theorem: $UD = \{\langle M \rangle \mid M \text{ is a TM that does } \textbf{not} \text{ accept on input } \langle M \rangle\}$ is undecidable

Proof: Suppose for contradiction that TM D decides UD

A more useful undecidable language

$A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts input } w\}$

Theorem: A_{TM} is undecidable

Proof: Assume for the sake of contradiction that TM H decides A_{TM}:

$$H(\langle M, w\rangle) = \begin{cases} \text{accept} & \text{if } M \text{ accepts } w \\ \text{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

Idea: Show that H can be used to construct a decider for the (undecidable) language UD -- a contradiction.

A more useful undecidable language

$A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts input } w\}$

Proof (continued):

Suppose, for contradiction, that H decides A_{TM}

Consider the following TM D:

"On input $\langle M\rangle$ where M is a TM:

1. Run H on input $\langle M, \langle M\rangle\rangle$
2. If H accepts, reject. If H rejects, accept."

Claim: D decides $UD = \{\langle M\rangle \mid \text{TM } M \text{ does not accept } \langle M\rangle\}$

...but this language is undecidable!

Unrecognizable Languages

Theorem: A language L is decidable if and only if L and $\overline{L}$ are both Turing-recognizable.

Corollary: $\overline{A_{\mathrm{TM}}}$ is unrecognizable

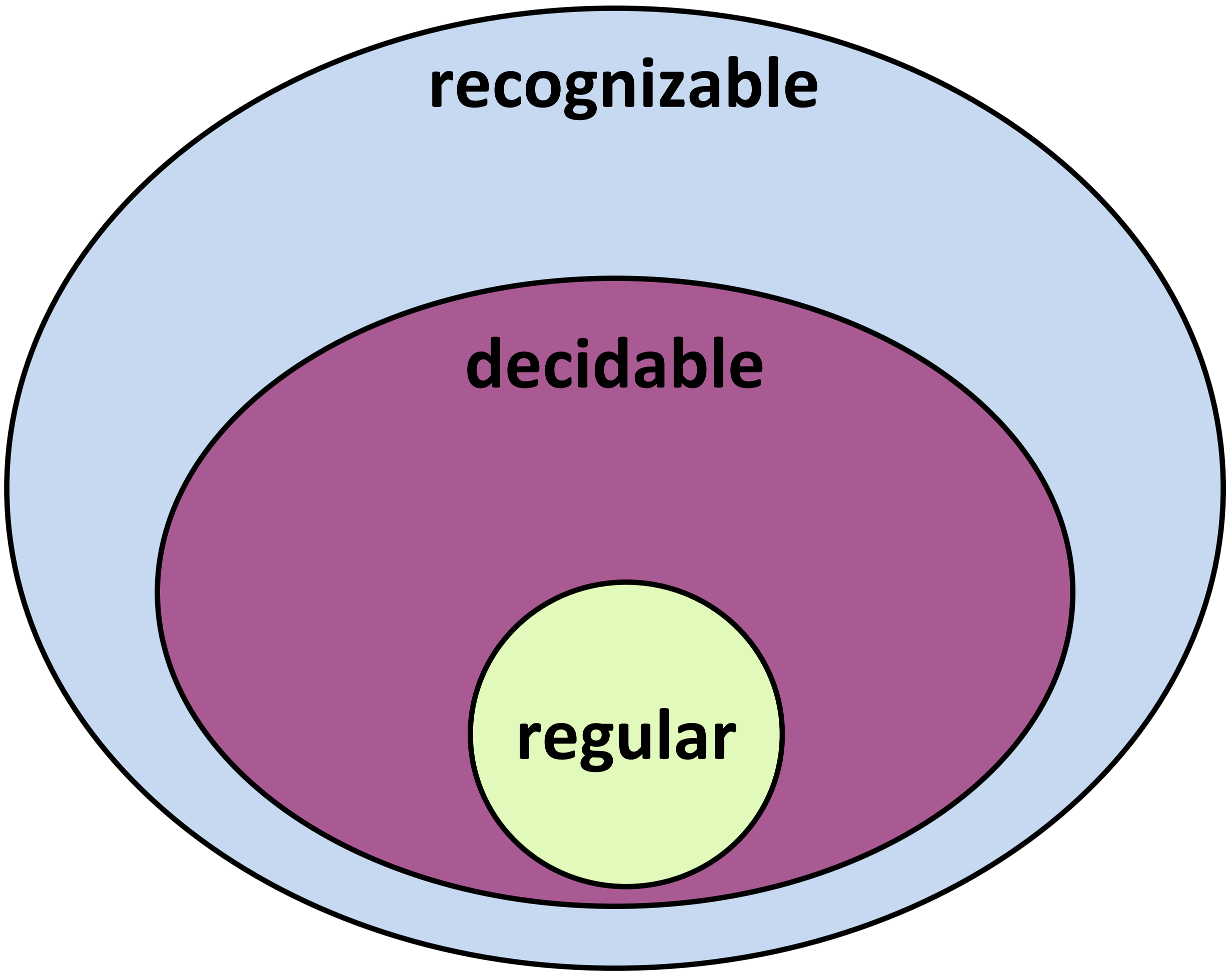
Proof of Theorem:

Unrecognizable Languages

Theorem: A language L is decidable if and only if L and $\overline{L}$ are both Turing-recognizable.

Proof continued:

Classes of Languages



Reductions

Scientists vs. Engineers

A computer scientist and an engineer are stranded on a desert island. They find two palm trees with one coconut on each. The engineer climbs a tree, picks a coconut and eats.

The computer scientist climbs the second tree, picks a coconut, climbs down, climbs up the first tree and places it there, declaring success.

"Now we've reduced the problem to one we've already solved."

(Please laugh)

Reductions

A reduction from problem A to problem B is an algorithm solving problem A which uses an algorithm solving problem B as a subroutine

If such a reduction exists, we say "A reduces to B"

Reductions

A reduction from problem A to problem B is an algorithm solving problem A which uses an algorithm solving problem B as a subroutine

If such a reduction exists, we say "A reduces to B"

If A reduces to B, and B is decidable, what can we say about A?

a) A is decidable

b) A is undecidable

c) A might be either decidable or undecidable

Two uses of reductions

Positive uses: If A reduces to B and B is decidable, then A is also decidable

$EQ_{\mathrm{DFA}} = \{\langle D_1, D_2\rangle \mid D_1, D_2 \text{ are DFAs and } L(D_1) = L(D_2)\}$

Theorem: EQ_{DFA} is decidable

Proof: The following TM decides EQ_{DFA}

On input $\langle D_1, D_2\rangle$, where $\langle D_1, D_2\rangle$ are DFAs:

1. Construct a DFA D that recognizes the symmetric difference $L(D_1) \triangle L(D_2)$
2. Run the decider for E_{DFA} on $\langle D\rangle$ and return its output

Two uses of reductions

Negative uses: If A reduces to B and A is undecidable, then B is also undecidable

$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM that accepts input } w\}$
Suppose H decides A_{TM}

Consider the following TM D.

On input $\langle M \rangle$ where M is a TM:
1. Run H on input $\langle M, \langle M \rangle \rangle$
2. If H accepts, reject. If H rejects, accept.

Claim: D decides

$$UD = \{\langle M \rangle \mid M \text{ is a TM that does not accept input } \langle M \rangle\}$$

Two uses of reductions

Negative uses: If A reduces to B and A is undecidable, then B is also undecidable

Template for undecidability proof by reduction:

1. Suppose to the contrary that B is decidable
2. Using a decider for B as a subroutine, construct an algorithm deciding A
3. But A is undecidable. Contradiction!

Halting Problem

Computational problem: Given a program (TM) and input w, does that program halt (either accept or reject) on input w?

Formulation as a language:

$$HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that halts on input } w\}$$

Ex. M = "On input x (a natural number written in binary):

For each $y = 1, 2, 3, \ldots$:

If $y^2 = x$, accept. Else, continue."

Is $\langle M, 101\rangle \in HALT_{\mathrm{TM}}$?

a) Yes, because M accepts on input 101

b) Yes, because M rejects on input 101

c) No, because M rejects on input 101

d) No, because M loops on input 101

Halting Problem

Computational problem: Given a program (TM) and input w, does that program halt (either accept or reject) on input w?

Formulation as a language:

$$HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that halts on input } w\}$$

Ex. M = "On input x (a natural number in binary):

For each $y = 1, 2, 3, \ldots$:

If $y^2 = x$, accept. Else, continue."

M' = "On input x (a natural number in binary):

For each $y = 1, 2, 3, \ldots, x$:

If $y^2 = x$, accept. Else, continue.

Reject."

Halting Problem

$$HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that halts on input } w\}$$

Theorem: $HALT_{\mathrm{TM}}$ is undecidable

Proof: Suppose for contradiction that there exists a decider H for $HALT_{\mathrm{TM}}$. We construct a decider for V for A_{TM} as follows:

On input $\langle M, w\rangle$:

1. Run H on input $\langle M, w\rangle$
2. If H rejects, reject
3. If H accepts, run M on w
4. If M accepts, accept

   Otherwise, reject.

This is a reduction from A_{TM} to $HALT_{\mathrm{TM}}$

Halting Problem

Computational problem: Given a program (TM) and input w, does that program halt on input w?

- A central problem in formal verification
- Dealing with undecidability in practice:
  - Use heuristics that are correct on most real instances, but may be wrong or loop forever on others
  - Restrict to a "non-Turing-complete" subclass of programs for which halting is decidable
  - Use a programming language that lets a programmer specify hints (e.g., loop invariants) that can be compiled into a formal proof of halting