BU CS 332 – Theory of Computation

https://forms.gle/GmpaSseFf4DfeHw78

Lecture 15:

- Undecidable and Unrecognizable Languages
- Reductions

Reading:

Sipser Ch 4.2, 5.1

Mark Bun

March 24, 2025

Undecidability

Last time: Countability, uncountability, and diagonalization

Existential proof that there are undecidable and unrecognizable languages

Today: An explicit undecidable language

Reductions: Relate decidability / undecidability of different problems

Specializing the proof

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

1) Consider the function $L: X \to P(\{0,1\}^*)$

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 UD is undecidable, hence L is not onto

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

Proof: Suppose for contradiction that some 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

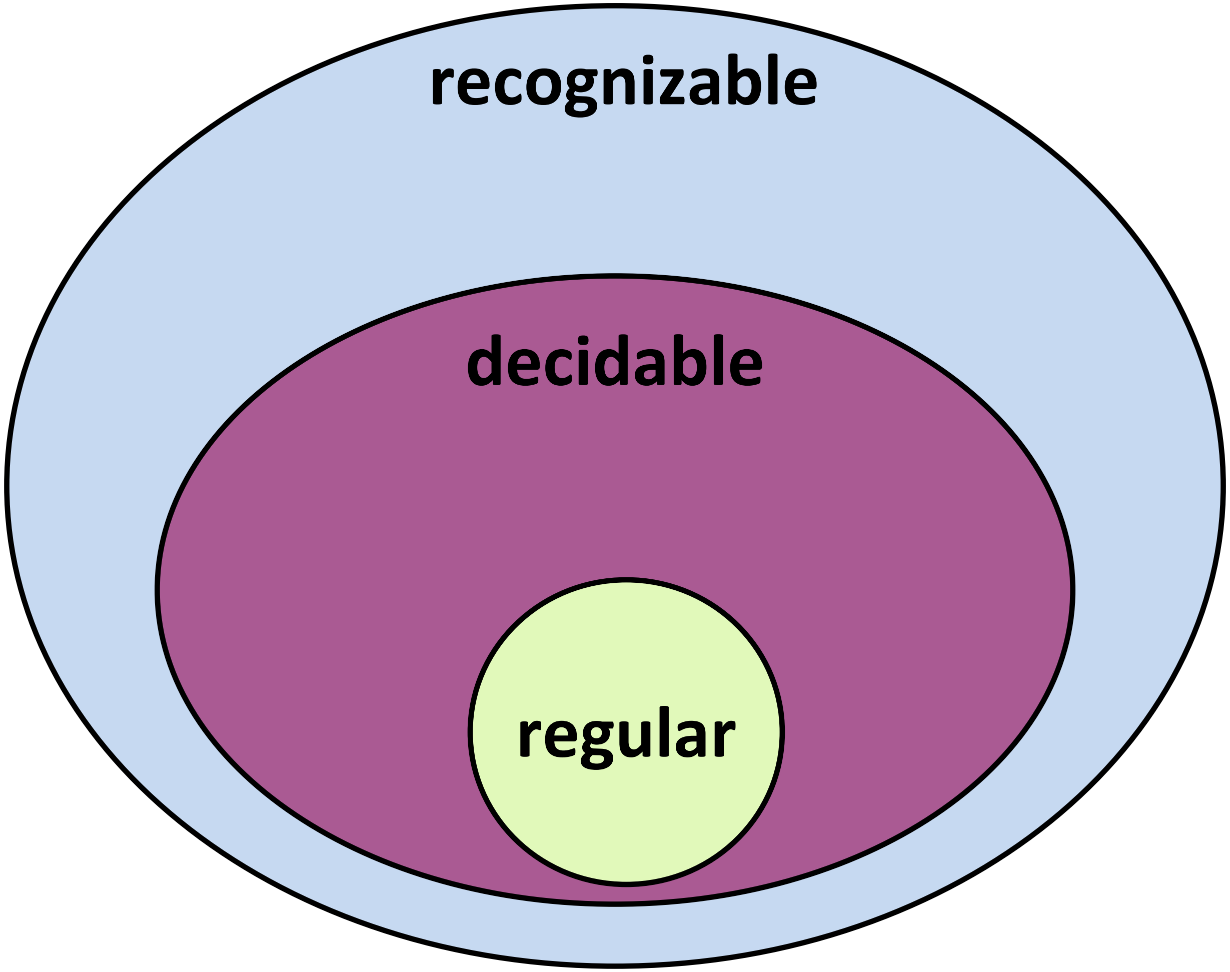
Proof of Theorem:

Unrecognizable Languages

Theorem: A language L is decidable if and only if L and $\bar{L}$ 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"

Two uses of reductions

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

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

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

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

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

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

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:

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

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

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

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

Proof: Suppose for contradiction that there exists a decider H for $HALT_{\text{TM}}$. 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 $HALT_{\text{TM}}$

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

Emptiness testing for TMs

$$E_{\mathrm{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

Theorem: E_{TM} is undecidable

Proof: Suppose for contradiction that there exists a decider R for E_{TM}. We construct a decider V for A_{TM} as follows:

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

1. Run R on input ???

This is a reduction from A_{TM} to E_{TM}

Emptiness testing for TMs

$$E_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

Theorem: E_{TM} is undecidable

Proof: Suppose for contradiction that there exists a decider R for E_{TM}. We construct a decider V for A_{TM} as follows:

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

1. Construct a TM N as follows:

2. Run R on input $\langle N \rangle$

3. If R , accept. Otherwise, reject

What do we want out of machine N?

a) $L(N)$ is empty iff M accepts w
b) $L(N)$ is non-empty iff M accepts w
c) $L(M)$ is empty iff N accepts w
d) $L(M)$ is non-empty iff N accepts w

This is a reduction from A_{TM} to E_{TM}

Emptiness testing for TMs

$$E_{\mathrm{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

Theorem: E_{TM} is undecidable

Proof: Suppose for contradiction that there exists a decider R for E_{TM}. We construct a decider V for A_{TM} as follows:

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

1. Construct a TM N as follows:

   "On input x:

   Run M on w and output the result."

2. Run R on input $\langle N \rangle$
3. If R rejects, accept. Otherwise, reject

This is a reduction from A_{TM} to E_{TM}

Interlude: Formalizing Reductions (Sipser 6.3)

Informally: A reduces to B if a decider for B can be used to construct a decider for A

One way to formalize:

- An *oracle* for language B is a device that can answer questions "Is $w \in B$?"
- An *oracle TM* M^B is a TM that can query an oracle for B in one computational step

A is Turing-reducible to B (written $A \leq_T B$) if there is an oracle TM M^B deciding A