BU CS 332 – Theory of Computation

https://forms.gle/DF9Ew4AyisH3Dy419

Lecture 15:

- More on Reductions

Reading:

Sipser Ch 5.1

All discussion sections will run tomorrow

Mark Bun

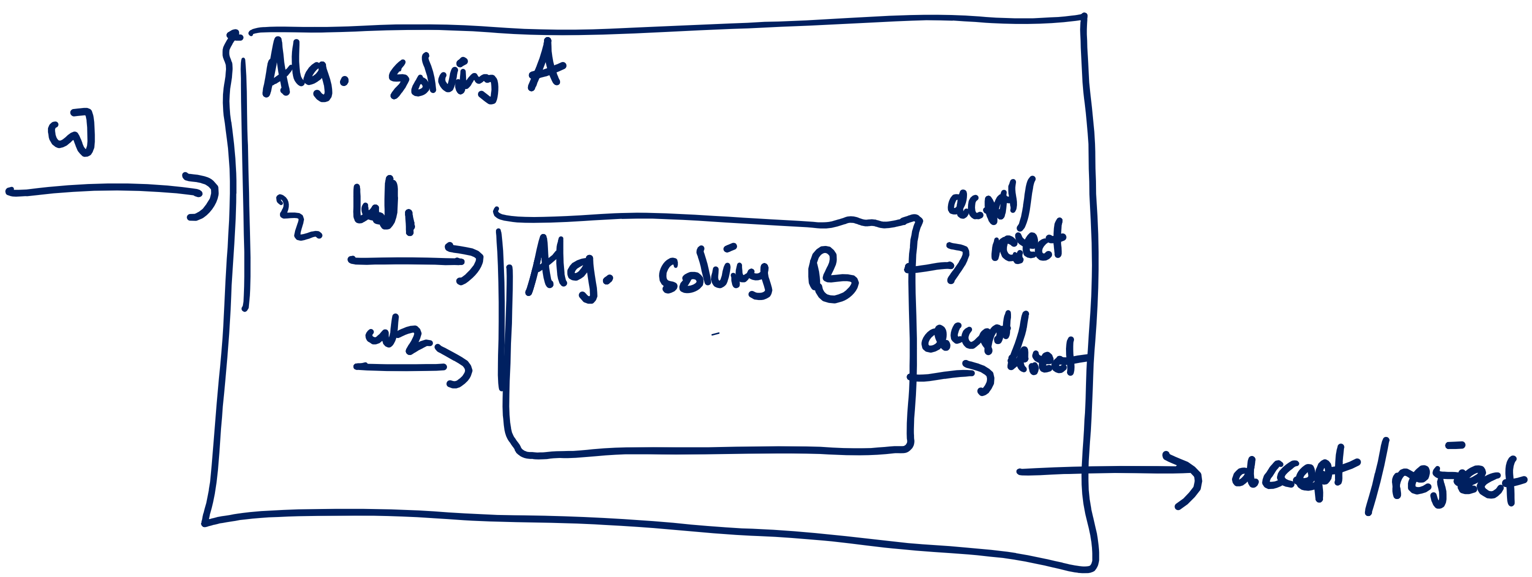
March 25, 2024

Reductions

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}

$E_{DFA} = \{\langle D \rangle \mid L(D) = \emptyset\}$

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}

Know: UD undecidable

$\Downarrow$ reduction from UD to A_{TM}

Conclude: A_{TM} also undecidable

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:

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

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."

binary encoding of 5

$101 \notin L(M)$ M loops forever on 101

Is $\langle M, \overbrace{101}^{\text{binary encoding of } 5}\rangle \in HALT_{\mathrm{TM}}$?

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."

$\langle M, 101\rangle \notin HALT_{TM}$

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."

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

$\forall x$

$\langle M', x\rangle \in HALT_{TM}$

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:

TM V

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.

Preprocess $\langle M, w\rangle$ to determine whether M halts on w

Given that M halts on w, can just run M on w and take its answer

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

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.

Claim: If H decides $HALT_{TM}$, then V decides A_{TM}.

Case 1: $\langle M, w\rangle \in A_{TM} \Rightarrow M$ accepts w
- In step 1, H accepts since M halts on w
- Go on to step 3, where M accepts w
- V accepts overall

Case 2: $\langle M, w\rangle \notin A_{TM}$

Subcase a: M rejects w
- In step 1, H accepts since M halts on w
- Go on to step 3 where M rejects w
- V rejects overall.

Subcase b: M loops on w
- In step 1, H rejects since $\langle M, w\rangle \notin HALT_{TM}$
- V rejects overall

Know A_{TM} is undecidable, so $\times$.
$\Rightarrow$ Our assumption $\exists H$ deciding $HALT_{TM}$ was false.
$\Rightarrow$ $HALT_{TM}$ undecidable

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

Emptiness testing for TMs

Computational problem: Given encoding of a TM M, determine whether M recognizes the empty language

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

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. Run R on input ???

Run R on input $\langle M \rangle$

- If M accepts w, then $w \in L(M)$, so $L(M) \neq \emptyset$ $\Rightarrow R$ rejects $\langle M \rangle$.
- If M does not accept w
  M might recognize $\emptyset \Rightarrow R$ accept $\langle M \rangle$
  or M accepts some string $w' \Rightarrow R$ rejects $\langle M \rangle$

If R rejected $\langle M \rangle$:

Don't know if M accepts w or not!

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:

$L(N) \neq \emptyset$

$\Leftrightarrow$ R rejects $\langle N \rangle$

$\Leftrightarrow$ V accepts $\langle M, w \rangle$ ($\Leftrightarrow$ M accepts w)

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

3. If R rejects, 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

• If M accepts w, then N accepts everything
• If M does not accept w then N does not accept anything

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

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:
   placeholder variable for input to N
   "On input x: Ignoring x
   Run M on w and output the result."
2. Run R on input $\boxed{\langle N \rangle}$
3. If R rejects, accept. Otherwise, reject

Claim: $L(N) \neq \emptyset$ iff M accepts w

Proof:

1) M accepts w:
$L(N) = \{x \mid M \text{ accepts } w\}$
$= \Sigma_1^* \neq \emptyset$

2) M does not accept w:
$L(N) = \{x \mid M \text{ accepts } w\}$
$= \emptyset$

⇒ If E_{TM} decidable, then A_{TM} decidable
But A_{TM} undecidable ↯
⇒ E_{TM} undecidable

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

Equality Testing for TMs

Recall: we showed EQ_{DFA} is decidable via a reduction from EQ_{DFA} to E_{DFA}.

$$EQ_{\text{TM}} = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Theorem: EQ_{TM} is undecidable

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

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

On input $\langle M \rangle$:

1. Construct TMs N_1, N_2 as follows:

   N_1 = N_2 =

2. Run R on input $\langle N_1, N_2 \rangle$
3. If R accepts, accept. Otherwise, reject.

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

Equality Testing for TMs

What do we want out of the machines N_1, N_2?

a) $L(M) = \emptyset$ iff $N_1 = N_2$ b) $L(M) = \emptyset$ iff $L(N_1) = L(N_2)$

c) $L(M) = \emptyset$ iff $N_1 \neq N_2$ d) $L(M) = \emptyset$ iff $L(N_1) \neq L(N_2)$

Goal: $\langle N_1, N_2 \rangle \in EQ_{TM} \iff L(N_1) = L(N_2) \iff L(M) = \emptyset \iff \langle M \rangle \in E_{TM}$

On input $\langle M \rangle$:

1. Construct TMs N_1, N_2 as follows:

N_1 = N_2 =

Suffices: $L(N_1) = \emptyset$ $L(N_2) = L(M)$

2. Run R on input $\langle N_1, N_2 \rangle$
3. If R accepts, accept. Otherwise, reject.

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

Equality Testing for TMs

$$EQ_{\mathrm{TM}} = \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$$

Theorem: EQ_{TM} is undecidable

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

TM V

On input $\langle M\rangle$:

1. Construct TMs N_1, N_2 as follows:

   N_1 = "On input x:
   reject"

   $N_2 = M$

2. Run R on input $\langle N_1, N_2\rangle$
3. If R accepts, accept. Otherwise, reject.

Analysis:

Claim: V accepts $\langle M\rangle \iff \langle M\rangle \in E_{TM}$

$\langle M\rangle \in E_{TM} \iff$
$L(M) = \emptyset \iff$
$L(N_1)\ (= \emptyset) = L(M)$
$= L(N_2)$
$\iff \langle N_1, N_2\rangle \in EQ_{TM}$
$\iff R$ accepts
$\iff V$ accepts.

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

Regular language testing for TMs

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

Theorem: REG_{TM} is undecidable

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

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

1. Construct a TM N as follows:

   Goal: construct N such that

   M accepts w => L(N) is regular

   M does not accept w => L(N) is nonregular

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

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

Regular language testing for TMs

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

Theorem: REG_{TM} is undecidable

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

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

1. Construct a TM N as follows:

   N = "On input x,

   1. If $x \in \{0^n 1^n \mid n \geq 0\}$, accept
   2. Run TM M on input w
   3. If M accepts, accept. Otherwise, reject."

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

Analysis:

- M accepts w $\Rightarrow$ $L(N) = \{0,1\}^*$ (regular) $\Rightarrow$ R accepts $\Rightarrow$ decider overall accepts
- M does not accept w $\Rightarrow$ $L(N) = \{0^n 1^n \mid n \geq 0\}$ $\Rightarrow$ R rejects $\Rightarrow$ decider overall rejects.

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