BU CS 332 – Theory of Computation

https://forms.gle/DF9Ew4AyisH3Dy419

Lecture 16:

- More on Reductions

Reading:

Sipser Ch 5.1

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

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

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: 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."

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:

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

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

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

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

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

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

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

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:

   "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

Equality Testing for TMs

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

Theorem: EQ_{TM} is undecidable

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 $L(N_1) \neq L(N_2)$

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

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

Theorem: EQ_{TM} is undecidable

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

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.

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:

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

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

Other undecidable problems

Problems in Language Theory

Apparent dichotomy:

- TMs seem to be able to solve problems about the power of weaker computational models (e.g., DFAs)
- TMs can't solve problems about the power of TMs themselves

Question: Are there undecidable problems that do not involve TM descriptions?

A_{DFA} **decidable**	A_{TM} **undecidable**
E_{DFA} **decidable**	E_{TM} **undecidable**
EQ_{DFA} **decidable**	EQ_{TM} **undecidable**

Undecidability of mathematics [Sipser 6.2]

Peano arithmetic: Formalization of mathematical statements about the natural numbers, using $+, \times, \leq$

Ex: "There exist infinitely many primes"

Theorem [Church, Turing]:

$\mathrm{TPA} = \{ \langle \varphi \rangle \mid \varphi \text{ is a true statement in PA} \}$ is undecidable

Proof skeleton:

Gödel's First Incompleteness Theorem [Sipser 6.2]

Theorem: There exists a true statement φ in Peano arithmetic that is not provable

Proof idea:

Suppose for contradiction that every true statement is provable. Then $\mathrm{TPA} = \mathrm{PPA}$ where

$\mathrm{PPA} = \{\langle\varphi\rangle \mid \varphi \text{ is a } provable \text{ statement in PA}\}$

Claim: PPA is Turing-recognizable

Construct a decider for TPA as follows:

A simple undecidable problem

<u>Post Correspondence Problem (PCP) [Sipser 5.2]:</u>

Domino: $\left[\frac{a}{ab}\right]$. Top and bottom are strings.

Input: Collection of dominos.

$$\left[\frac{aa}{aba}\right], \left[\frac{ab}{aba}\right], \left[\frac{ba}{aa}\right], \left[\frac{abab}{b}\right]$$

Match: List of some of the input dominos (repetitions allowed) where top = bottom

$$\left[\frac{ab}{aba}\right], \left[\frac{aa}{aba}\right], \left[\frac{ba}{aa}\right], \left[\frac{aa}{aba}\right], \left[\frac{abab}{b}\right]$$

Problem: Does a match exist? This is undecidable

Computation History Method

A sequence of configurations $C_0, \ldots, C_\ell$ is an accepting computation history for TM M on input w if

1. C_0 is the start configuration $q_0 w_1 \ldots w_n$
2. Every C_{i+1} legally follows from C_i
3. C_ℓ is an accepting configuration

Reduction from the undecidable language A_{TM} to a language L using the following idea:

Given an input $\langle M, w \rangle$ to A_{TM}, the ability to solve L enables checking whether there exists an accepting computation history for M on w