Lecture 16:
• More on Reductions

Reading:
Sipser Ch 5.1

Mark Bun
November 3, 2022
Reductions
Reductions

A **reduction** from problem \( A \) to problem \( B \) is an algorithm for problem \( A \) which uses an algorithm for problem \( B \) as a subroutine.

If such a reduction exists, we say “\( A \) reduces to \( B \)”

**Positive uses:** If \( A \) reduces to \( B \) and \( B \) is decidable, then \( A \) is also decidable.

Ex. \( E_{DFA} \) is decidable \( \Rightarrow \) \( EQ_{DFA} \) is decidable

**Negative uses:** If \( A \) reduces to \( B \) and \( A \) is undecidable, then \( B \) is also undecidable.

Ex. \( UD \) is undecidable \( \Rightarrow \) \( A_{TM} \) is undecidable
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:
\[
HALT_{TM} = \{ \langle M, w \rangle | M \text{ is a TM that halts on input } w \} 
\]

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

\[
\text{For each } y = 1, 2, 3, \ldots : \\
\quad \text{If } y^2 = x, \text{ accept. Else, continue.}
\]

Is \( \langle M, 101 \rangle \in HALT_{TM} \)?

a) Yes, because \( M \) accepts on input 101
b) Yes, because \( M \) rejects on input 101
c) No, because \( M \) rejects on input 101
\boxed{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:

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

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

\[ \text{HALT}_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that halts on input } w \} \]

Theorem: \( \text{HALT}_{TM} \) is undecidable

Proof: Suppose for contradiction that there exists a decider \( H \) for \( \text{HALT}_{TM} \). 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 \( \text{HALT}_{TM} \).
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

\[ E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \]

**Theorem:** \( E_{TM} \) is undecidable

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

On input \( \langle M, w \rangle \):

1. Run \( R \) on input \( \langle M, w \rangle \)

\[ M_2 = \text{On input } x : \]
   - \( \text{If } x = w : \text{ run } M \text{ on } w \) \text{ and reject.} \]
   - \( \text{Else : reject.} \)

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:** \( E_{\text{TM}} \) is undecidable

**Proof:** Suppose for contradiction that there exists a decider \( R \) for \( E_{\text{TM}} \). We construct a decider for \( A_{\text{TM}} \) as follows:

On input \( \langle M, w \rangle \):

1. Construct a TM \( N \) as follows:
   
   \[
   \begin{align*}
   &\text{If } \langle M, w \rangle \in A_{\text{TM}} \Rightarrow L(N) \neq \emptyset \Rightarrow R \text{ rejects } \langle N \rangle \\
   &\text{If } \langle M, w \rangle \notin A_{\text{TM}} \Rightarrow L(N) = \emptyset \Rightarrow R \text{ accepts } \langle N \rangle 
   \end{align*}
   \]
   
   \( \Rightarrow \text{decider accepts} \)

2. Run \( R \) on input \( \langle N \rangle \)

3. If \( R \text{ rejects} \), 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 \( A_{\text{TM}} \) to \( E_{\text{TM}} \)
Emptiness testing for TMs

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

**Theorem:** \( E_{TM} \) is undecidable

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

On input \( \langle M, w \rangle \):

1. Construct a TM \( N \) as follows:
   - "On input \( x \): Ignore \( x \)
     - 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

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

**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:
   
   \[ N_1 = \quad \quad \quad N_2 = \]

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 = \text{"on input } x:\text{ reject"}
   - $N_2 = M$
   - $L(N_1) = \emptyset$
   - $L(N_2) = L(M)$

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

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

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:
   \[ N_1 = \quad N_2 = \]
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 \( EQ_{TM} \)
Regular language testing for TMs

\[ \text{REG}_{\text{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular} \} \]

**Theorem:** \( \text{REG}_{\text{TM}} \) is undecidable

**Proof:** Suppose for contradiction that there exists a decider \( R \) for \( \text{REG}_{\text{TM}} \). We construct a decider for \( A_{\text{TM}} \) 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 \( A_{\text{TM}} \) to \( \text{REG}_{\text{TM}} \)
Regular language testing for TMs

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

**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:
   \[ N = \text{“On input } x, \]
   \[ 1. \text{ If } x \in \{0^n1^n \mid n \geq 0\}, \text{ accept} \]
   \[ 2. \text{ Run TM } M \text{ on input } w \]
   \[ 3. \text{ If } M \text{ accepts, accept. Otherwise, reject.”} \]
2. Run \( R \) on input \( \langle N \rangle \)
3. If \( R \) accepts, accept. Otherwise, reject

This is a reduction from \( A_{TM} \) to \( REG_{TM} \)