Mapping Reducibility

Wednesday, April 19, 2023
Announcements

• HW 9 still out
  • Due Sun 4/23 11:59pm EST

Quiz Preview

• Mapping reducibility is a relation between two ...?
Last time: Undecidable ...

- $\text{REGULAR}_{\text{TM}} = \{<M> | M \text{ is a TM and } L(M) \text{ is a regular language}\}$

- $\text{CONTEXTFREE}_{\text{TM}} = \{<M> | M \text{ is a TM and } L(M) \text{ is a CFL}\}$

- $\text{DECIDABLE}_{\text{TM}} = \{<M> | M \text{ is a TM and } L(M) \text{ is a decidable language}\}$

- $\text{FINITE}_{\text{TM}} = \{<M> | M \text{ is a TM and } L(M) \text{ is a finite language}\}$

- ...

- $\text{ANYTHING}_{\text{TM}} = \{<M> | M \text{ is a TM and “… anything …” about } L(M)\}$
Flashback: “Reduced”

Thm: $HALT_{TM}$ is undecidable

Proof, by contradiction:

- Assume: $HALT_{TM}$ has decider $R$; use it to create $A_{TM}$ decider:

  $S = \text{“On input } \langle M, w \rangle \text{, an encoding of a TM } M \text{ and a string } w:\n  \begin{align*}
  1. \text{ Run TM } R \text{ on input } \langle M, w \rangle . \\
  2. \text{ If } R \text{ rejects, reject.} \\
  3. \text{ If } R \text{ accepts, simulate } M \text{ on } w \text{ until it halts.} \\
  4. \text{ If } M \text{ has accepted, accept; if } M \text{ has rejected, reject.”}
  \end{align*}

  A potential problem: could the conversion itself go into an infinite loop?

  Let’s formalize this conversion, i.e., mapping reducibility
Flashback: $A_{NFA}$ is a decidable language

$A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts input string } w \}$

Decider for $A_{NFA}$:

$N = \text{"On input } \langle B, w \rangle, \text{ where } B \text{ is an NFA and } w \text{ is a string:"

1. Convert NFA $B$ to an equivalent DFA $C$, using the procedure $\text{NFA} \to \text{DFA}$
2. Run TM $M$ on input $\langle C, w \rangle$.
3. If $M$ accepts, accept; otherwise, reject."

We said this NFA $\to$ DFA algorithm is a decider TM, but it doesn’t accept/reject?

More generally, our analogy has been: “programs $\sim$ TMs”, but programs do more than accept/reject?
Definition: Computable Functions

A function \( f : \Sigma^* \rightarrow \Sigma^* \) is a **computable function** if some Turing machine \( M \), on every input \( w \), halts with just \( f(w) \) on its tape.

• A **computable function** is represented with a TM that, instead of accept/reject, “outputs” its final tape contents

• Example 1: All arithmetic operations

• Example 2: Converting between machines, like DFA→NFA
  • E.g., adding states, changing transitions, wrapping TM in TM, etc.
Definition: Mapping Reducibility

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the **reduction** from $A$ to $B$.

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.
Flashback: Equivalence of Contrapositive

“If $X$ then $Y$” is equivalent to ...

1. “If $Y$ then $X$” (converse)

2. “If not $X$ then not $Y$” (inverse)

3. “If not $Y$ then not $X$” (contrapositive)
Flashback: Equivalence of Contrapositive

“If $X$ then $Y$” is equivalent to ... ?

× “If $Y$ then $X$” (converse)
  • No!

× “If not $X$ then not $Y$” (inverse)
  • No!

✓ “If not $Y$ then not $X$” (contrapositive)
  • Yes!
Definition: Mapping Reducibility

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the **reduction** from $A$ to $B$.

- **“forward” direction** ($\Rightarrow$): if $w \in A$ then $f(w) \in B$
- **“reverse” direction** ($\Leftarrow$): if $f(w) \in B$ then $w \in A$

Equivalent (contrapositive): if $w \notin A$ then $f(w) \notin B$

Reverse direction just as important: don’t convert non-As into Bs

Easier to prove
Proving Mapping Reducibility: 2 Steps

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a **computable function** $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the **reduction** from $A$ to $B$.

**Step 1:** Show there is computable fn $f$... by creating a TM

**Step 2:** Prove the iff is true for that computable fn TM

**Step 2a:** “forward” direction ($\Rightarrow$): if $w \in A$ then $f(w) \in B$

**Step 2b:** “reverse” direction ($\Leftarrow$): if $f(w) \in B$ then $w \in A$

**Step 2b, alternate (contrapositive):** if $w \notin A$ then $f(w) \notin B$

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.
**Thm:** $A_{TM}$ is mapping reducible to $\text{HALT}_{TM}$

To show: $A_{TM} \leq_{m} \text{HALT}_{TM}$

**Step 1:** create computable fn $f$: $<M, w> \rightarrow <M', w>$ where:

Step 2: show $<M, w> \in A_{TM}$ if and only if $<M', w> \in \text{HALT}_{TM}$

The following machine $F$ computes a reduction $f$.

$F =$ “On input $<M, w>$:

1. Construct the following machine $M'$.
   $M' =$ “On input $x$:
   1. Run $M$ on $x$.
   2. If $M$ accepts, accept.
   3. If $M$ rejects, enter a loop.”
2. Output $<M', w>$.”

$M'$ is like $M$, except it always loops when it doesn’t accept

Output new $M'$

**Step 2:**

$M$ accepts $w$

if and only if

$M'$ halts on $w$
⇒ If $M$ accepts $w$, then $M'$ halts on $w$

- $M'$ accepts (and thus halts) if $M$ accepts

⇐ If $M'$ halts on $w$, then $M$ accepts $w$

---

The following machine $F$ computes a reduction $f$.

$F =$ “On input $<M, w>$:

1. Construct the following machine $M'$.
   $M'$ = “On input $x$:
   1. Run $M$ on $x$.
   2. If $M$ accepts, accept.
   3. If $M$ rejects, enter a loop.”

2. Output $<M', w>$.”

---

Step 2: $M$ accepts $w$ if and only if $M'$ halts on $w$
⇒ If \( M \) accepts \( w \), then \( M' \) halts on \( w \)
  • \( M' \) accepts (and thus halts) if \( M \) accepts

⇐ If \( M' \) halts on \( w \), then \( M \) accepts \( w \)

⇐ (Alternatively) If \( M \) doesn’t accept \( w \), then \( M' \) doesn’t halt on \( w \) (contrapositive)
  • Two possibilities for “doesn’t accept”:
    1. \( M \) loops: \( M' \) loops and doesn’t halt
    2. \( M \) rejects: \( M' \) loops and doesn’t halt

The following machine \( F \) computes a reduction \( f \).
\[
F = \text{“On input } \langle M, w \rangle \text{:}
\]
1. Construct the following machine \( M' \).
   \( M' = \text{“On input } x \text{:}
   \]
   1. Run \( M \) on \( x \).
   2. If \( M \) accepts, accept.
   3. If \( M \) rejects, enter a loop.”
2. Output \( \langle M', w \rangle \).”

Step 2:
\( M \) accepts \( w \) if and only if \( M' \) halts on \( w \)

If \( M \) loops, then \( M' \) loops

If \( M \) rejects, then \( M' \) loops

Now we know what mapping reducibility is, and how to prove it for two languages; but what is it used for?
Uses of Mapping Reducibility

• To prove Decidability

• To prove Undecidability
Thm: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

**Proof** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$N = \text{"On input } w:\n1. \text{ Compute } f(w).\n2. \text{ Run } M \text{ on input } f(w) \text{ and output whatever } M \text{ outputs."}$

Why is it true that: If $M$ accepts $f(w)$ then $N$ should accept $w$?? i.e., $f(w)$ in $B$ guarantees that $w$ in $A$??

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$ 

The function $f$ is called the **reduction** from $A$ to $B$. 

We know this is true bc of the iff (specifically the reverse direction)
Corollary: If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.

- **Proof** by contradiction.
- **Assume** $B$ is decidable.
- **Then** $A$ is decidable (by the previous thm).
- **Contradiction:** we already said $A$ is undecidable.
Summary: Showing Mapping Reducibility

Language \( A \) is **mapping reducible** to language \( B \), written \( A \leq_m B \), if there is a **computable function** \( f: \Sigma^* \rightarrow \Sigma^* \), where for every \( w \),

\[ w \in A \iff f(w) \in B. \]

The function \( f \) is called the **reduction** from \( A \) to \( B \).

**Step 1:**
Show there is computable fn \( f \) ... by creating a TM

**Step 2:**
Prove the \( \iff \) is true

**Step 2a:** “forward” direction (\( \Rightarrow \)): if \( w \in A \) then \( f(w) \in B \)

**Step 2b:** “reverse” direction (\( \Leftarrow \)): if \( f(w) \in B \) then \( w \in A \)

**Step 2b, alternate (contrapositive):** if \( w \notin A \) then \( f(w) \notin B \)

A function \( f: \Sigma^* \rightarrow \Sigma^* \) is a **computable function** if some Turing machine \( M \), on every input \( w \), halts with just \( f(w) \) on its tape.
Summary: Using Mapping Reducibility

To prove decidability...

- If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

To prove undecidability...

- If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.

Be careful with the direction of the reduction, i.e., what is known and what is unknown!
Alternate Proof: The Halting Problem

\[ \text{HALT}_{TM} \text{ is undecidable} \]

- If \( A \leq_m B \) and \( A \) is undecidable, then \( B \) is undecidable.
- \( A_{TM} \leq_m \text{HALT}_{TM} \)
- Since \( A_{TM} \) is undecidable,
- \( \ldots \) and we showed mapping reducibility from \( A_{TM} \) to \( \text{HALT}_{TM} \),
- then \( \text{HALT}_{TM} \) is undecidable \( \blacksquare \)
Flashback: \( EQ_{\text{TM}} \) is undecidable

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

Proof by contradiction:

- **Assume** \( EQ_{\text{TM}} \) has **decider** \( R \); use it to create **\( E_{\text{TM}} \) decider:**

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

\[ S = \text{“On input } \langle M \rangle \text{, where } M \text{ is a TM:} \]

1. Run \( R \) on input \( \langle M, M_1 \rangle \), where \( M_1 \) is a TM that rejects all inputs.
   
2. If \( R \) accepts, accept; if \( R \) rejects, reject.”
Alternate Proof: $EQ_{TM}$ is undecidable

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

Show mapping reducibility: $E_{TM} \leq_m EQ_{TM}$

Step 1: create computable fn $f$: $<M> \rightarrow <M_1, M_2>$, computed by $S$

$S =$ “On input $\langle M \rangle$, where $M$ is a TM:
1. Construct: $\langle M, M_1 \rangle$, where $M_1$ is a TM that rejects all inputs.
2. Output: $\langle M, M_1 \rangle$

Step 2: show iff requirements of mapping reducibility (hw exercise)

And use theorem ...

Undecidability Proof Technique #4: Mapping Reducibility + theorem

If $A \leq_m B$ and $A$ is undecidable, then $B$ is undecidable.
Flashback: \( E_{TM} \) is undecidable

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

Proof, by contradiction:

• Assume \( E_{TM} \) has decider \( R \); use it to create \( A_{TM} \) decider:

\[ S = \text{"On input } \langle M, w \rangle, \text{ an encoding of a } \text{TM } M \text{ and a string } w:\]

1. Use the description of \( M \) and \( w \) to construct the \( \text{TM } M_1 \)

2. Run \( R \) on input \( \langle M_1 \rangle \).

3. If \( R \) accepts, reject; if \( R \) rejects, accept."

• So this only reduces \( A_{TM} \) to \( \overline{E_{TM}} \)

If \( M \) accepts \( w \), then \( M_1 \) accepts \( w \), meaning \( M_1 \) is not in \( E_{TM} \)!
Alternate Proof: $E_{TM}$ is undecidable

Show mapping reducibility??: $A_{TM} \leq_m E_{TM}$

Step 1: create computable fn $f$: $<M, w> \rightarrow <M'>$,
computed by $S$

1. Use the description of $M$ and $w$ to construct the TM $M_1$

2. Output: $<M_1>$.

3. If $R$ accepts, reject; if $R$ rejects, accept.”

So this only reduces $A_{TM}$ to $\overline{E_{TM}}$.

It’s good enough! Still proves $E_{TM}$ is undecidable

- If ... undecidable langs are closed under complement

Step 2: show iff requirements of mapping reducibility
(hw exercise)
Language Complement

**Complement** (OPPO from hw3) of a language $A$, written $\overline{A}$ ...

... is the set of all strings not in set $A$

*Example:*

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

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

$\bigcup \{ w \mid w \text{ is a string that is not a TM description} \}$
Undecidable Langs Closed under Complement

Proof by contradiction

• Assume some lang \( L \) is undecidable and \( \overline{L} \) is decidable ...
  • Then \( \overline{L} \) has a decider

• ... then we can create decider for \( L \) from decider for \( \overline{L} \) ...
  • Because decidable languages are closed under complement (hw10?)!
Next Time: Turing Unrecognizable?

Is there anything out here?

Where do these undecidable languages go?

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

\[ EQ_{CFG} = \{ \langle G, H \rangle | G \text{ and } H \text{ are CFGs and } L(G) = L(H) \} \]

\[ EQ_{TM} = \{ \langle M_1, M_2 \rangle | M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \} \]
Check-in Quiz 4/19

On gradescope