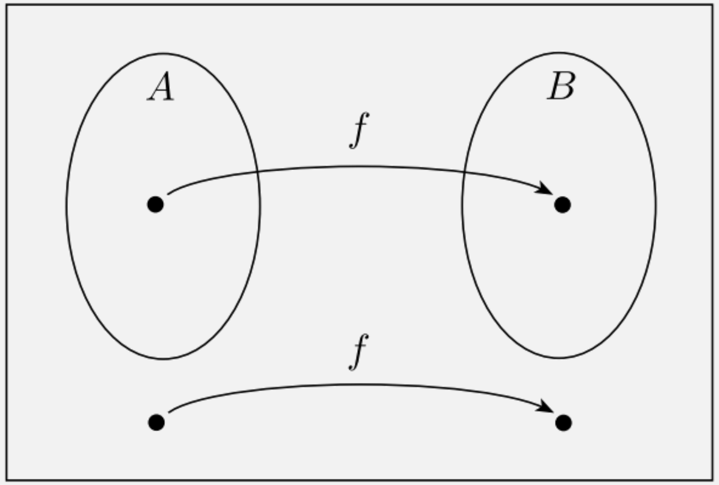
UMB CS 420

Mapping Reducibility

Monday, April 4, 2022



Announcements

- HW 8 extended
  - Due Wed 4/6 11:59pm EST

- HW 9 out soon

Last Time: TM Accepting Computations

A TM **accepting computation** is sequence of configurations, where:

1. Start Config:
   - State: **start state,**
   - Head: **at leftmost cell**
   - Tape: **has input string**

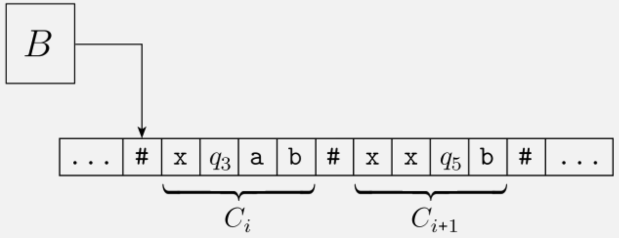
“w”

2. End Config:
   - State: **accept state**

So: **any machine that can recognize TM accepting sequences …**

… can be used to **implement A_{TM} decider!**

i.e., **…** can be used to **prove undecidability!**



3. Middle Configs:
   - State + Head + Tape: **each step must be valid according to δ**

Last Time: What Makes CFLs "Context-Free"?

- $A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ Decidable
- $E_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ ← Why is this **decidable?** Decidable
- $ALL_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$ ← But this is **undecidable?** **Undecidable**

This unintuitive result is explained by …

… the fact that PDAs **can** recognize **non-accepting** TM config sequences

Can be computed in a "context-free" way:
check that pairs of configs are valid nondeterministically,
… and accept if **any** are not

… but PDAs **cannot** recognize **accepting** TM config sequences

Cannot be computed in a "context-free" way:
check that pairs of configs are valid nondeterministically,
… and accept if **all** are not

This gives insight into what makes context-free languages "context-free"

The Post Correspondence Problem (PCP)

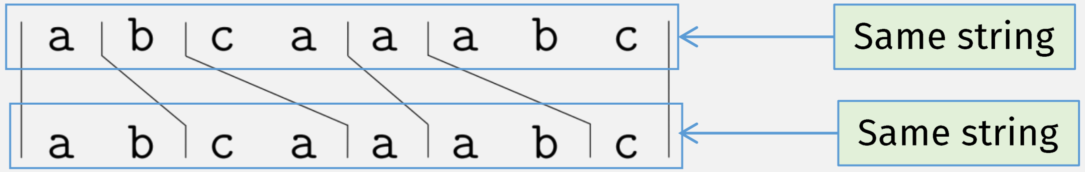
A unique undecidable problem

A Non-Formal Languages Undecidable Problem: *PCP*

- Let P be a set of "**dominos**" $\left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$
  - Where each t_i and b_i are strings
  - E.g., $P = \left\{ \left[\frac{\mathtt{b}}{\mathtt{ca}}\right], \left[\frac{\mathtt{a}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right] \right\}$
- A **match** is:
  - A sequence of dominos with the same top and bottom strings

Repeats allowed

  - E.g., $\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{b}}{\mathtt{ca}}\right]\left[\frac{\mathtt{ca}}{\mathtt{a}}\right]\left[\frac{\mathtt{a}}{\mathtt{ab}}\right]\left[\frac{\mathtt{abc}}{\mathtt{c}}\right]$ ➡



- Then: PCP = { <P> | P is a set of dominos with a match }

Theorem: PCP is undecidable

PCP = { <P> | P is a set of dominos with a match }

Proof by contradiction:

Assume PCP is decidable, has decider R; use it to create decider for A_{TM}:

On input <M, w>:
1. Construct a set of dominos P that has a **match** only when M accepts w
2. Run R with P as input
3. Accept if R accepts, else reject

So a match is a sequence of configs showing M accepting w!

Idea: P has M's TM configurations as its domino strings

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

PCP Dominos

- First domino: $\left[\frac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right]$ **Start config** (on bottom)
- Key idea: add dominos representing valid TM steps:

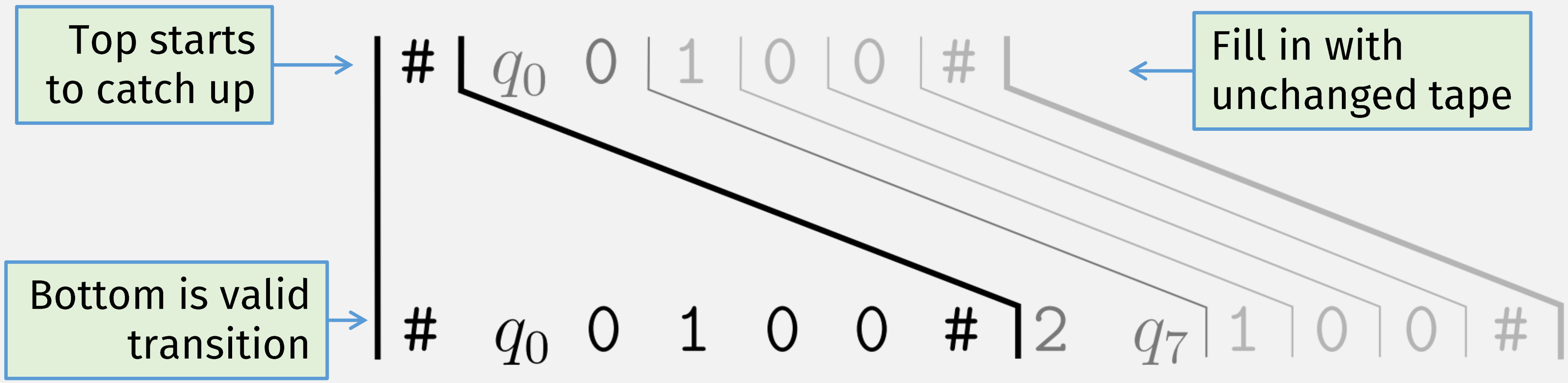
$$\text{if } \delta(q,a) = (r,b,\mathrm{R}), \text{ put } \left[\frac{qa}{br}\right] \text{ into } P$$

$$\text{if } \delta(q,a) = (r,b,\mathrm{L}), \text{ put } \left[\frac{cqa}{rcb}\right] \text{ into } P$$

- For the tape cells that don't change: $\text{put } \left[\frac{a}{a}\right] \text{ into } P$
- Top can only "catch up" if there is an accepting config sequence

PCP Example

- Let w = `0100` and $\delta(q_0, 0) = (q_7, 2, \mathrm{R})$ so $\left[\frac{q_0 0}{2 q_7}\right]$ in P



PCP Dominos (accepting)

- When accept state reached, let top "catch" up:

For every $a \in \Gamma$,

put $\left[\frac{a\, q_{\text{accept}}}{q_{\text{accept}}}\right]$ and $\left[\frac{q_{\text{accept}}\, a}{q_{\text{accept}}}\right]$ into P

Bottom "eats" one char

Only possible match:
accepting sequence of TM configs

"eat" one char

Mapping Reducibility

Flashback: "Reduced"

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ known

$\Downarrow$

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

Thm: $HALT_{\mathsf{TM}}$ is undecidable

Proof, by contradiction:

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

... into an A_{TM} string

$S =$ "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

Essentially, **we** convert a hypothetical $HALT_{\mathsf{TM}}$ string ...

Use R to first check if M will loop on w

Then run M on w knowing it won't loop

- Contradiction: A_{TM} is undecidable and has no decider!

Let's formalize this conversion, i.e., **mapping reducibilty**

Flashback: A_{NFA} is a decidable language

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

Decider for A_{NFA} :

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

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

We said this **NFA→DFA** algorithm is a TM, but it doesn't accept/reject?

More generally, we've been saying
"**programs = TMs**",
but programs do more than accept/reject?

Definition: Computable Functions

- Has TM that, instead of accept/reject, "outputs" final tape contents

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

- 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

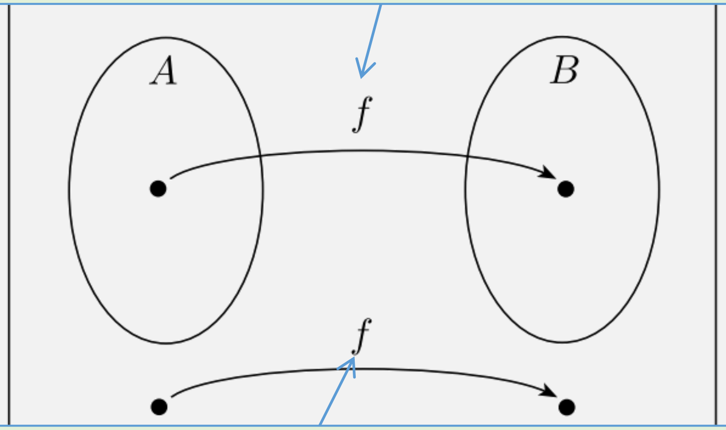
Language A is ***mapping reducible*** to language B, written $\boxed{A \leq_{\mathrm{m}} B,}$ if there is a $\boxed{\text{computable function } f\colon \Sigma^* \longrightarrow \Sigma^*}$, where for every w,

$$\boxed{w \in A \iff f(w) \in B.}$$

"if and only if"

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

"forward" direction (⇒): if $w \in A$ then $f(w) \in B$



"reverse" direction (⇐): if $f(w) \in B$ then $w \in A$

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ 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

$\boxed{\text{“If } X \text{ then } Y\text{”}}$ is equivalent to … ?

- “If Y then X” (converse)
  - No!
- “If not X then not Y” (inverse)
  - No!

✓ $\boxed{\text{“If not } Y \text{ then not } X\text{”}}$ (contrapositive)
  - **Yes!**

Definition: Mapping Reducibility

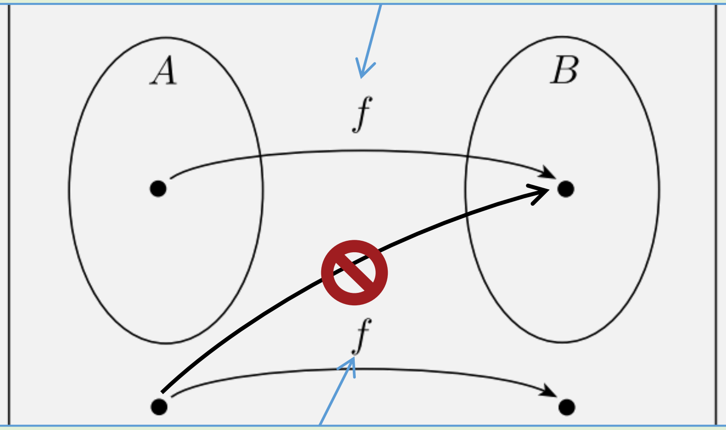
Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

$$\boxed{w \in A \iff f(w) \in B.}$$

"if and only if"

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 $f(w) \notin B$

Proving Mapping Reducibility: 2 Steps

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

Language A is ***mapping reducible*** to language B, written $A \leq_m B$, if there is a $\boxed{\text{computable function } f\colon \Sigma^* \longrightarrow \Sigma^*}$, where for every w,

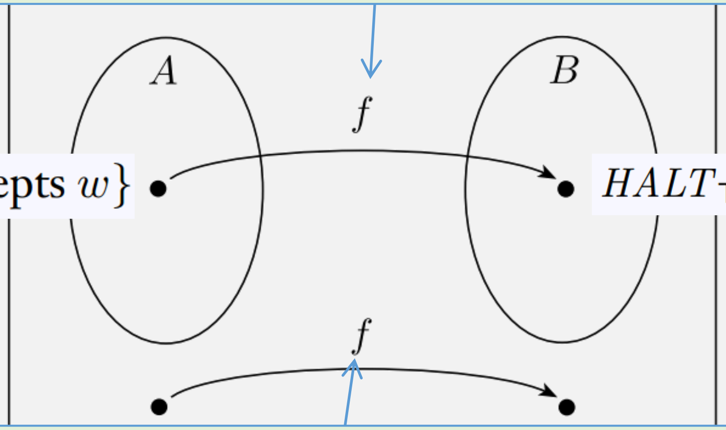
$$\boxed{w \in A \iff f(w) \in B.}$$

"if and only if"

Step 2:
Prove the iff is true

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

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



$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

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

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

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

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ 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 $HALT_{\mathsf{TM}}$

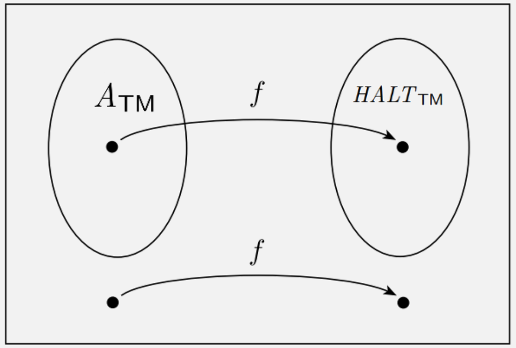
$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

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

- To show: $A_{\mathsf{TM}} \leq_{\mathrm{m}} HALT_{\mathsf{TM}}$

Step 1: computable fn *f*: <*M*, *w*> → <*M'*, *w*> where:

$$\langle M, w\rangle \in A_{\mathsf{TM}} \text{ if and only if } \langle M', w'\rangle \in HALT_{\mathsf{TM}}$$



The following machine F computes a reduction f.

F = "On input $\langle M, w\rangle$:
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 $\langle M', w\rangle$."

Converts *M* to *M'*

Step 2:
M accepts *w*
if and only if
M' halts on *w*

Output new *M'*

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

Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, 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 $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

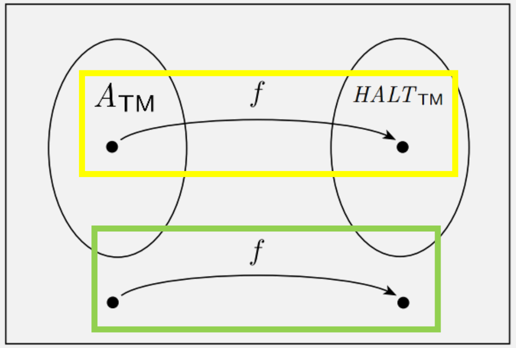
⇒ If M accepts w, then M' halts on w

- 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 non-acceptance:
  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 = "On input $\langle \boxed{M}, w\rangle$:

1. Construct the following machine M'.

   $\boxed{M'}$ = "On input x:

   1. Run M on x.
   2. If M accepts, *accept*.
   3. If M rejects, enter a loop."

2. Output $\langle M', w\rangle$."

Uses of Mapping Reducibility

- To prove **Decidability**

- To prove **Undecidability**

Thm: If $A \le_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Has a decider

Must create decider

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.

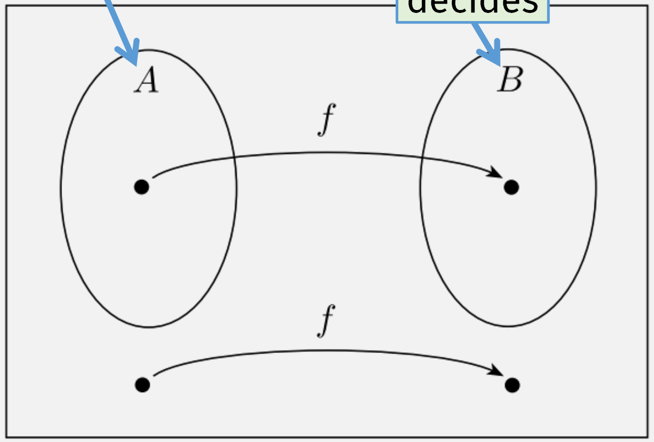
$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

decides

decides

We know this is true bc of the iff (specifically reverse direction)



Language A is ***mapping reducible*** to language B, written $A \le_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

$$\boxed{w \in A \iff f(w) \in B.}$$

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

Coro: If $A \leq_{\mathrm{m}} B$ 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

If $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Summary: Showing Mapping Reducibility

Step 1:
Show there is computable fn f … by creating a TM

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

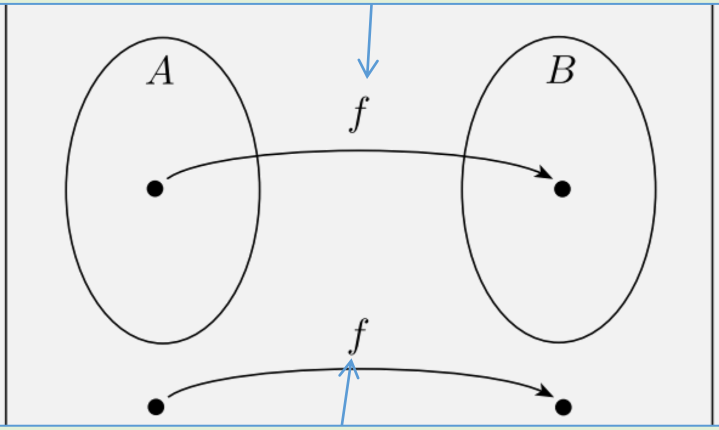
$$w \in A \iff f(w) \in B.$$

"if and only if"

Step 2:
Prove the iff is true

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

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

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ 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 $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Known

Unknown
(want to prove)

To prove undecidability …

- If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Be careful with the **direction of the reduction**!

Alternate Proof: The Halting Problem

$HALT_{\mathsf{TM}}$ is undecidable

- If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Must be known

- $A_{\mathsf{TM}} \leq_{\mathrm{m}} HALT_{\mathsf{TM}}$

- Since A_{TM} is undecidable,
- … and we showed mapping reducibility from A_{TM} to $HALT_{\mathsf{TM}}$,
- then $HALT_{\mathsf{TM}}$ is undecidable ■

Flashback: EQ_{TM} is undecidable

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

Proof by contradiction:

- Assume EQ_{TM} has *decider* R; use to create E_{TM} *decider*:

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

S = "On input $\langle M\rangle$, where M 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

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

Show **mapping reducibility:** $E_{\mathsf{TM}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$

Step 1: create computable fn f: $\langle M\rangle \rightarrow \langle M_1, M_2\rangle$, 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 (exercise)

And use theorem …

If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Flashback: E_{TM} is undecidable

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

Proof, by contradiction:

- Assume E_{TM} has *decider* R; use to create A_{TM} *decider*:

$S =$ "On input $\langle M, w\rangle$, an encoding of a TM M and a string w:

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

$M_1 =$ "On input x:

1. If $x \neq w$, *reject*.

2. If $x = w$, run M on input w and *accept* if M does."

2. Run R on input $\langle M_1 \rangle$.

3. If R accepts, *reject*; if R rejects, *accept*."

If M accepts w, M_1 not in E_{TM}!

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

Alternate Proof: E_{TM} is undecidable

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

Show **mapping reducibility??:** $A_{\mathsf{TM}} \leq_{\mathrm{m}} E_{\mathsf{TM}}$

Step 1: create computable fn *f*: <*M*, *w*> → <*M'*>, computed by *S*

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

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

$M_1 =$ "On input x:

1. If $x \neq w$, *reject*.

2. If $x = w$, run M on input w and *accept* if M does."

2. Output: $\langle M_1 \rangle$.

3. ~~If R accepts, *reject*; if R rejects, *accept*.~~"

If M accepts w, M_1 not in E_{TM}!

- So this only reduces A_{TM} to $\overline{E_{\mathsf{TM}}}$
- It's good enough! Still proves E_{TM} is undecidable
  - Because undecidable langs are closed under complement

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

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

Contradiction!

- ... then we can create decider for L from decider for $\overline{L}$ ...
  - Because **decidable languages are closed under complement** (hw8)!

Check-in Quiz 4/4

On gradescope