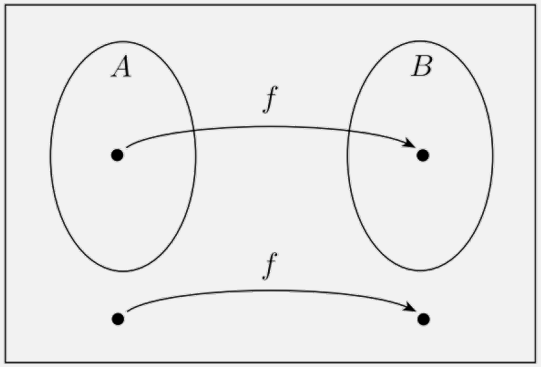
Mapping Reducibility, Metaprogramming

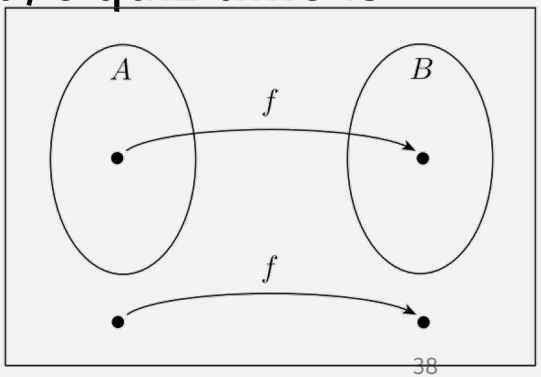


To Update

- HW8/9 out Fri sat – Due Wed right before break
- Thank earlier class, just make HW7 due on Thursday @ 10 before exam
- HW7 input output files slow, behind. I will add up _some_
  - Use JFLAP files posted + examples I give
  - Crowdsource examples and I/O examples as a way to prep for exam.
    - I bless this!
- No Reg Machines on this Exam :/// But on optional final
- Friday I'll record lecture; wed we'll do some exam review
- Autograder rety count for exam – 7 tries
- Exam Friday 8am to Sat 10pm (should not be 2x as long, just more flexible timing)
- Open extra NOT FOR CREDIT dropboxes for HW467

Announcements

- _Still_ behind on autograder, file generating
- W/o tester, I worry I'll hand you out buggy files.
- _Some_ exam q & a, now not @ end of class b/c quiz time is yours



Last time: "Reduced" A_{TM} to $HALT_{\text{TM}}$

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

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

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

Proof, by contradiction:

- Assume $HALT_{\text{TM}}$ 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. 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*."

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!

Today: Formalize "reduction" and "reducibilty"

Reducing to non-A_{TM} language

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

Thm: EQ_{TM} is undecidable

Proof, by contradiction:

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

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

$S =$ "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*."

Reducing to non-A_{TM} language

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

Thm: EQ_{TM} is undecidable

Proof, by contradiction:

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

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

$S =$ "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*."

- Contradiction: E_{TM} is undecidable!

Summary

- $A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ — Decidable
- $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ — Decidable
- $A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ — **Undecidable**
- $E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$ — Decidable
- $E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ — Decidable
- $E_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ — **Undecidable**
- $EQ_{\text{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ — Decidable
- $EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ — **Undecidable**
- $EQ_{\text{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$ — **Undecidable**

Observation:
Can we decide anything about Turing Machines, i.e., about programs?

Can't decide anything about TMs?

- $REGULAR_{TM} = \{<M> \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}$ **Undecidable**

HW9

- $CONTEXTFREE_{TM} = \{<M> \mid M \text{ is a TM and } L(M) \text{ is a CFL}\}$ **Undecidable**
- $DECIDABLE_{TM} = \{<M> \mid M \text{ is a TM and } L(M) \text{ is a decidable language}\}$ **Undecidable**
- $FINITE_{TM} = \{<M> \mid M \text{ is a TM and } L(M) \text{ is a finite language}\}$ **Undecidable**
- …

Undecidable:

Rice's Theorem

HW9

- $ANYTHING_{TM} = \{<M> \mid M \text{ is a TM and "something something" about } L(M)\}$

Today: Computable Functions

- Needed to formalize the notion of “reducibility”

Flashback: A_{NFA} is a decidable language

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

Decider (i.e., "run" function) 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 for this conversion given in Theorem 1.39.
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?

Computable Functions

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

DEFINITION 5.17

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.

Mapping Reducibility

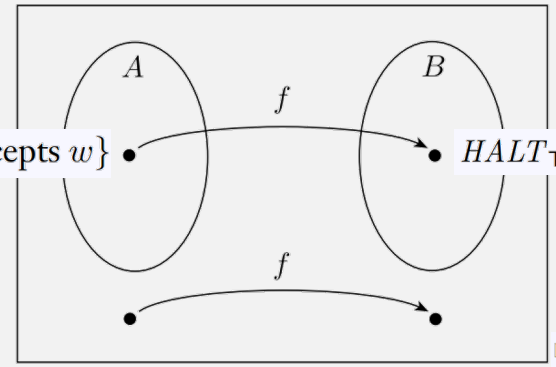
DEFINITION 5.20

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

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

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

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



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

DEFINITION 5.17

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.

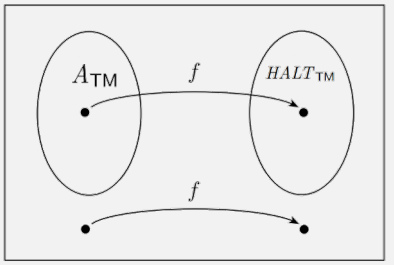
Thm: A_{TM} is mapping reducible to $HALT_{\text{TM}}$

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

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

- To show: $A_{\text{TM}} \leq_{\text{m}} HALT_{\text{TM}}$
- Want: computable fn f : $\langle M, w\rangle$ → $\langle M', w'\rangle$ where:

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



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'*

M accepts *w* if and only if *M'* halts on *w*

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

Output new *M'*

DEFINITION 5.20

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

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

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

DEFINITION 5.17

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.

How is mapping reducibility useful?

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

Has a decider

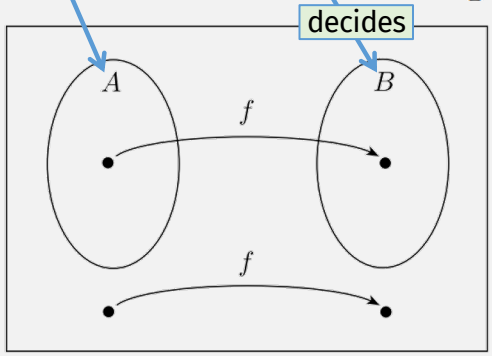
PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

N = "On input w:

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

decides

decides



DEFINITION 5.20

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

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

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

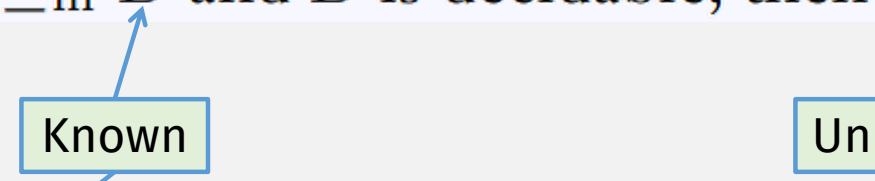
Coro: If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

- 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: Mapping Reducibility Theorems

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



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

Alternate Proof: The Halting Problem

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

- If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.
- $A_{\mathsf{TM}} \leq_{\mathrm{m}} HALT_{\mathsf{TM}}$
- Since A_{TM} is undecidable, then $HALT_{\mathsf{TM}}$ is undecidable

Alternate Proof: EQ_{TM} is undecidable

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

Flashback: proof by contradiction:

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

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

$S =$ "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: Show: $E_{\mathsf{TM}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$

- Computable fn f : $\langle M\rangle$ → $\langle M, M_1\rangle$

DEFINITION 5.20

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

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

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

Reducing to complement: E_{TM} is undecidable

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

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 just described.
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, *reject*; if R rejects, *accept*."

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."

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

Alternate proof: computable fn: $\langle M, w \rangle$ → $\langle M_1 \rangle$???

- So this only reduces A_{TM} to $\overline{E_{\mathsf{TM}}}$
- Still proves E_{TM} is undecidable, since undecidable langs closed under complement

More Helpful Theorems

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

If $A \leq_{\mathrm{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

- Same proofs as:

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

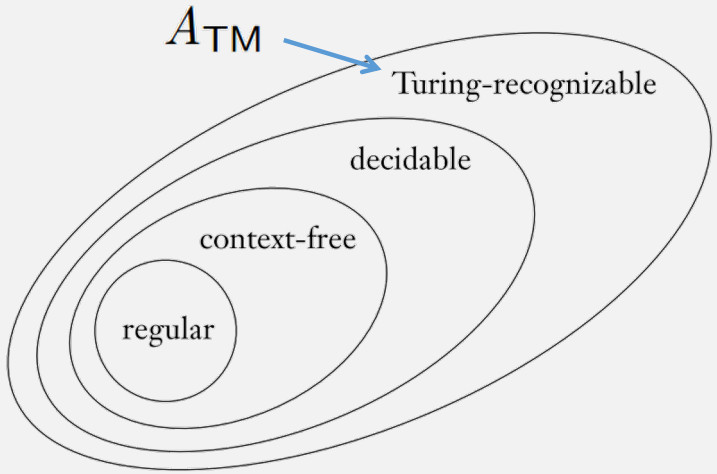
If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Thm: EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

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

1. EQ_{TM} is not Turing-recognizable

$\overline{A_{\mathsf{TM}}}$



$\overline{A_{\mathsf{TM}}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$ A is not Turing-recognizable, th EQ_{TM} not Turing-recognizable.

Mapping Reducibility implies Mapping Red. of Complements

DEFINITION 5.20

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

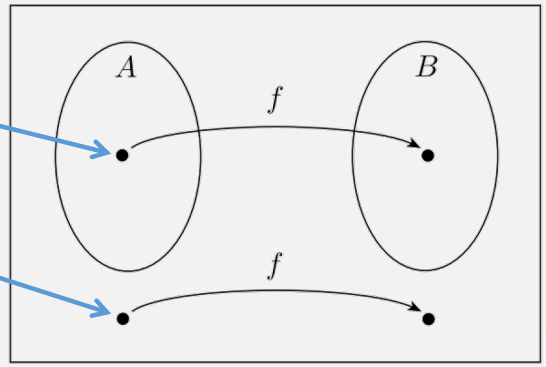
$$w \in A \iff f(w) \in B.$$

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

$A \leq_{\mathrm{m}} B$

implies

$\overline{A} \leq_{\mathrm{m}} \overline{B}$



DEFINITION 5.17

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.

Thm: EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

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

1. EQ_{TM} is not Turing-recognizable

   Two Choices:

   - Create Computable fn: $\overline{A_{\text{TM}}} \rightarrow EQ_{\text{TM}}$
   - Or Computable fn: $A_{\text{TM}} \rightarrow \overline{EQ_{\text{TM}}}$

Thm: EQ_{TM} is not Turing-recognizable

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

- Create Computable fn: A_{TM} → $\overline{EQ_{\mathsf{TM}}}$
- $\langle M, w\rangle$ → $\langle M_1, M_2\rangle$ M_1 and M_2 are TMs and $L(M_1) \neq L(M_2)$

F = "On input $\langle M, w\rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2.
   M_1 = "On any input: ← Accepts nothing
      1. *Reject*."
   M_2 = "On any input: ← Accepts nothing or everything
      1. Run M on w. If it accepts, *accept*."
2. Output $\langle M_1, M_2\rangle$."

- If M accepts w, M_1 not equal to M_2
- If M does not accept w, M_1 equal to M_2

Thm: EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

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

1. EQ_{TM} is not Turing-recognizable
   - Create Computable fn: $\overline{A_{\text{TM}}} \rightarrow EQ_{\text{TM}}$
   - Or Computable fn: $A_{\text{TM}} \rightarrow \overline{EQ_{\text{TM}}}$
   - **DONE!**
2. $\overline{EQ_{\text{TM}}}$ is not co-Turing-recognizable
   - (A lang is co-Turing-recog. if it is complement of Turing-recog. lang)

Prev: EQ_{TM} is not Turing-recognizable

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

- Create Computable fn: $A_{\mathsf{TM}} \rightarrow \overline{EQ_{\mathsf{TM}}}$
- $\langle M, w\rangle \rightarrow \langle M_1, M_2\rangle$ M_1 and M_2 are TMs and $L(M_1) \neq L(M_2)$

F = "On input $\langle M, w\rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2.
   M_1 = "On any input: ← Accepts nothing
      1. *Reject*."
   M_2 = "On any input: ← Accepts nothing or everything
      1. Run M on w. If it accepts, *accept*."
2. Output $\langle M_1, M_2\rangle$."

DONE!

Now: $\overline{EQ}_{\text{TM}}$ is not Turing-recognizable

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

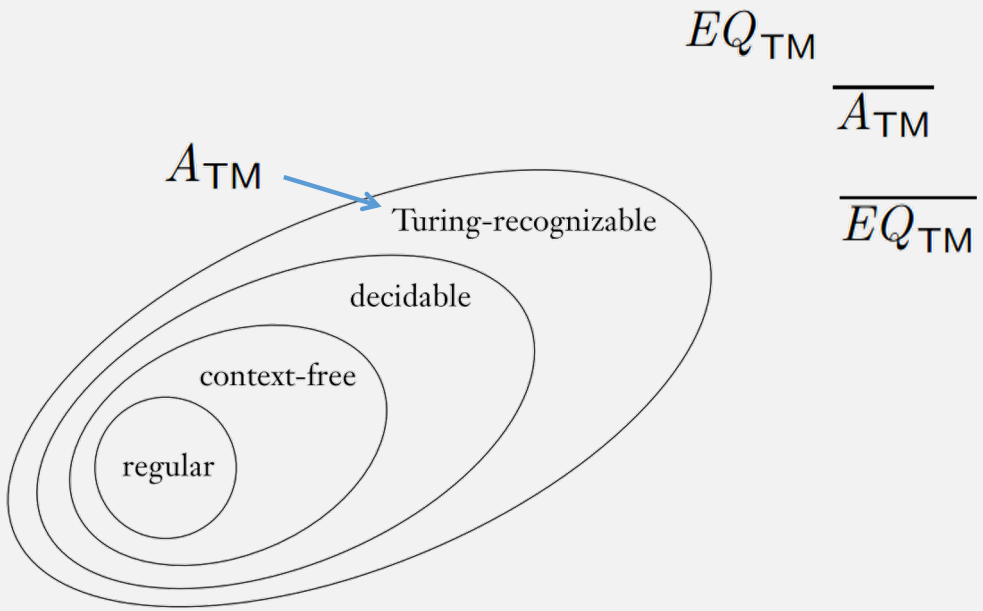
- Create Computable fn: A_{TM} → $\overline{EQ}_{\text{TM}}$
- $\langle M, w\rangle$ → $\langle M_1, M_2\rangle$ M_1 and M_2 are TMs and $L(M_1) \neq L(M_2)$

F = "On input $\langle M, w\rangle$, where M is a TM and w a string:

1. Construct the following two machines, M_1 and M_2.
   M_1 = "On any input: ← Accepts ~~nothing~~ everything
      1. *Accept*."
   M_2 = "On any input: ← Accepts nothing or everything
      1. Run M on w. If it accepts, *accept*."
2. Output $\langle M_1, M_2\rangle$."

DONE!

Unrecognizable Languages



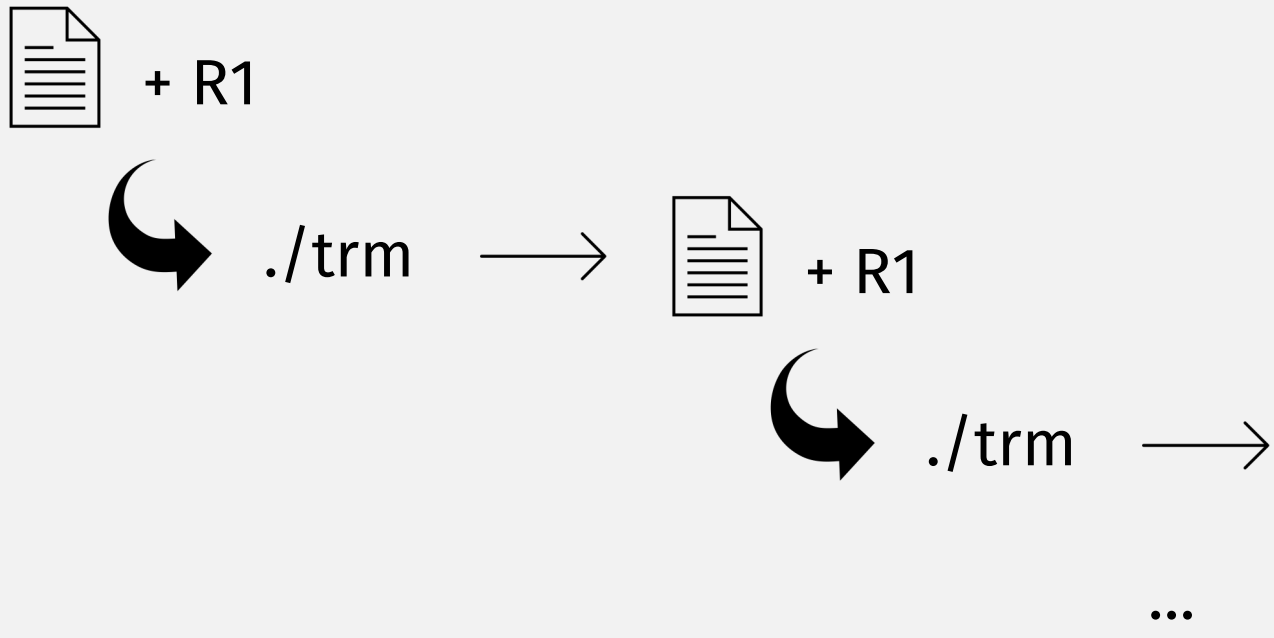
Register Machines II

Last Time

- TRM – 1# language. Simple instruction set
- Instructions are written in {1,#}. Data are {1,#}.
- Interact via command line operations, and we can see what happens

- φ_{program} : the function computed by that program

- Programs can take other programs as input or produce programs as output.


+ R1
./trm
+ R1
./trm
...

Check-in Quiz

On gradescope